Learning in Modular Systems

David M. Bradley

CMU-RI-TR-09-26

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

May 7, 2010

Thesis Committee
James A. Bagnell
Yoshua Bengio
Martial Hebert
Fernando De La Torre

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
07 MAY 2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Learning in Modular Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie MellonUniversity ,Robotics Institute,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Complex robotics systems are often built as a system of modules, where each module solves a separate data processing task to produce the complex overall behavior that is required of the robot. For instance, the perception system for autonomous o -road navigation discussed in this thesis uses a terrain classification module, a ground-plane estimation module, and a path-planning module among others. Splitting a complex task into a series of sub-problems allows human designers to engineer solutions for each sub-problem independently, and devise e cient specialized algorithms to solve them. However, modular design can also create problems for applying learning algorithms. Ideally, learning should find parameters for each module that optimize the performance of the overall system. This requires obtaining ?local? information for each module about how changing the parameters of that module will impact the output of the system. Previous work in modular learning [1, 2] showed that if the modules of system were di erentiable gradient descent could be used to provide this local information in ?shallow? systems containing with two or three modules between input and output. However, except for convolutional neural networks, this procedure was rarely successful in ?deep? systems of more than three modules. Many robotics applications added an additional complication by employing a planning algorithm to produce their output. This makes it hard to define a ?loss? function to judge how well the system is performing, or compute a gradient with respect to previous modules in the system. Recent advances in learning deep neural networks [3, 4] suggest that learning in deep systems can be successful if data-dependent regularization is first used to provide relevant local information to the modules of the system, and the modules are then jointly optimized by gradient descent. Concurrently, research in imitation learning [5, 6] has o ered e ective new ways of defining loss functions for the output of planning modules. This thesis combines these lines of research to develop new tools for learning in modular systems. As data-dependent regularization has been shown to be critical to success in deep modular systems, several significant contributions are provided in this area. A novel, di erentiable formulation of sparse coding is presented and shown to be a powerful semi-supervised learning algorithm. Sparse coding has traditionally used non-convex optimization methods, and an alternative, convex formulation is developed with a deterministic optimization procedure. Theoretical contributions developed for this convex formulation also enable an e cient, online multi-task learning algorithm.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**139**	

To my wife Janine, the love of my life.

Abstract

Complex robotics systems are often built as a system of modules, where each module solves a separate data processing task to produce the complex overall behavior that is required of the robot. For instance, the perception system for autonomous off-road navigation discussed in this thesis uses a terrain classification module, a ground-plane estimation module, and a path-planning module among others. Splitting a complex task into a series of sub-problems allows human designers to engineer solutions for each sub-problem independently, and devise efficient specialized algorithms to solve them. However, modular design can also create problems for applying learning algorithms. Ideally, learning should find parameters for each module that optimize the performance of the overall system. This requires obtaining "local" information for each module about how changing the parameters of that module will impact the output of the system.

Previous work in modular learning [1, 2] showed that if the modules of system were differentiable, gradient descent could be used to provide this local information in "shallow" systems containing with two or three modules between input and output. However, except for convolutional neural networks, this procedure was rarely successful in "deep" systems of more than three modules. Many robotics applications added an additional complication by employing a planning algorithm to produce their output. This makes it hard to define a "loss" function to judge how well the system is performing, or compute a gradient with respect to previous modules in the system.

Recent advances in learning deep neural networks [3, 4] suggest that learning in deep systems can be successful if *data-dependent* regularization is first used to provide relevant local information to the modules of the system, and the modules are then jointly optimized by gradient descent. Concurrently, research in imitation learning [5, 6] has offered effective new ways of defining loss functions for the output of planning modules.

This thesis combines these lines of research to develop new tools for learning in modular systems. As data-dependent regularization has been shown to be critical to success in deep modular systems, several significant contributions are provided in this area. A novel, differentiable formulation of sparse coding is presented and shown to be a powerful semi-supervised learning algorithm. Sparse coding has traditionally used non-convex optimization methods, and an alternative, convex formulation is developed with a deterministic optimization procedure. Theoretical contributions developed for this convex formulation also enable an efficient, online multi-task learning algorithm. Results in domain adaptation provide further regularization options. To allow joint optimization of systems that employ planning modules, this thesis leverages loss functions developed in recent imitation learning research, and develops techniques for improving all modules of the system with subgradient descent. Finally, this thesis has also made significant contributions to mobile robot per-

ception for navigation, providing terrain classification techniques that been incorporated into fielded industrial and government systems [7].

Acknowledgements

This thesis would not have been possible without the generous help given by many people over the last six years. I owe an enormous debt of gratitude to my advisor Drew Bagnell. In addition to guiding me through the Ph.D. program and teaching me about machine learning, Drew has been a great advisor and a pleasure to work for. My thesis committee members, Yoshua Bengio, Martial Hebert, and Fernando De la Torre have been generous with their time, and their comments and feedback have greatly improved this thesis. Several of my fellow graduate students have provided invaluable assistance. Dave Silver has provided help and insight on many projects. Aaron Morris has been a constant source of help and guidance throughout the last six years, and I would also like to thank my officemates Jean-Francois Lalonde, and Hongwen Kang for keeping me company on the long road to the thesis defense. I am also grateful to Tom Howard who went through the defense process slightly ahead of me, and helped to guide me through the process as well as providing help with planning algorithms and visualization graphics.

It has been a wonderful privilege to get to work on a state-of-the-art robot like crusher, and I would like to thank the UPI team that made that possible. Particularly Cris Dima, Carl Wellington, Cliff Olmsted, Tom Pilarski, and Constantine Domashev. I would also like to thank my masters's advisor, Scott Thayer, who introduced me to the field of robotics and guided me to the Ph.D. program. David Wettergreen, Nicolas Vandapel, and Alyosha Efros also deserve special thanks for their help in advising me during my masters and Ph.D.

Finally, I would not be where I am today without the support of my family. My parents, Mark and Mary, encouraged my interests as I was growing up, and put up with unfinished robotics projects littering the basement. It was great to be able to turn to my sister Sarah and my brother-in-law Erik, grad students themselves, for understanding during the dark period in grad school where the light at the beginning of the tunnel is far away, but the light at the end of the tunnel is not yet visible. Most importantly, though, my wife Janine has been a rock of encouragement, and I would like to thank her not only for getting me through the tough times when experiments and proofs were not working, but also for making grad school such an enjoyable time.

This work was made possible through several grants. The Army Research Office provided generous funding through an NDSEG Fellowship. Much of the work on perception for mobile robot navigation was provided by the Advanced Research Projects Agency (DARPA) under contract Unmanned Ground Combat Vehicle PerceptOR Integration (UPI, contract numberMDA972-01-9-0005). Most recently this work has been sponsored by the U.S. Army Research Laboratory, under contract Robotics Collaborative Technology Alliance (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be

interpreted as representing the official policies or endorsements of the the U.S. Government.

Table of Contents

Chapter 1

Introduction

Complex problems are often easier to solve when they are broken down into a set of sub-problems, and each part is addressed separately. This reductionist approach is a key technique in engineering, where it has been used to design nearly all aspects of modern life, everything from bridges to microprocessors. For instance, a new car is developed by specifying what is required of the components–wheels, seats, axel etc..–and how they will be combined to form the car. It is natural that learning problems should benefit from this approach as well. However, most current learning theory is instead devoted to the opposite goal: creating *black-box* learning methods that offer performance guarantees, and shield the designer as much as possible from having to interact with the internal structure and parameters of the algorithm. In the black-box paradigm, the *target task* that the system is supposed to learn is defined by a set of labeled examples, and a loss function that measures how well the system performed on those examples. This approach has had many successes, and has produced several algorithms including Nearest Neighbors, 2-layer Neural Networks and Support Vector Machines that are *universal approximators*, i.e. theoretically capable of solving any prediction problem given enough labeled training data. However, for several important complex learning problems encountered in robotics, it is infeasible to try to learn from data alone; in these cases incorporating human insight about the structure of a good solution is crucial for success.

This thesis adopts a reductionist approach to learning, in which human insight is first used to design a network of heterogeneous modules for solving the target task, and then learning is used to optimize the parameters of those modules. Learning in modular systems involves solving challenges in regularization and credit assignment. First, we argue that prior work in *deep learning* indicates that *data-dependent* regularization is critical to success in modular systems. Then several algorithms for providing data-dependent regularization are proposed and evaluated including contributions to sparse coding, multi-task learning, and domain adaptation. Finally, the credit assignment problem is examined, and approaches for learning with non-differentiable planning modules are investigated.

1.1 Deep Modular Systems

Prior work in machine learning learning has mainly dealt with algorithms that have *shallow* architectures, such as Nearest Neighbors, 2-layer Neural Networks and Support Vector Machines, where there are only two or three non-linear functions between the input data and the output prediction.

As mention previously, many of these shallow algorithms are theoretically capable of solving any prediction problem given enough labeled training data.

However, in practice these algorithms have serious limitations.

As is discussed in Section 2.2.1, shallow learning algorithms define a distance function between examples, and predict the class of a test example, based on the labels associated with "nearby" labeled points. If the target function varies rapidly with respect to the algorithm's distance function, then the algorithm will require many labeled examples in order to learn the task, as well as corresponding increases in storage and computation. Hence, it is critically important to pick a combination of input representation, internal distance function, and output representation that makes the target function vary slowly. In other words, a slowly varying function is easy to remember.

Functions with high-dimensional inputs and high-dimensional structured outputs can be very hard to memorize. In high dimensional spaces, everything is naturally far apart, and the distance function of the learner has to be carefully constructed in order to bring examples with the same class close enough together that the correct function can be learned. With structured output spaces, the output representation of learning must also be carefully constructed so that it is easy for the learning algorithm to generate valid predictions.

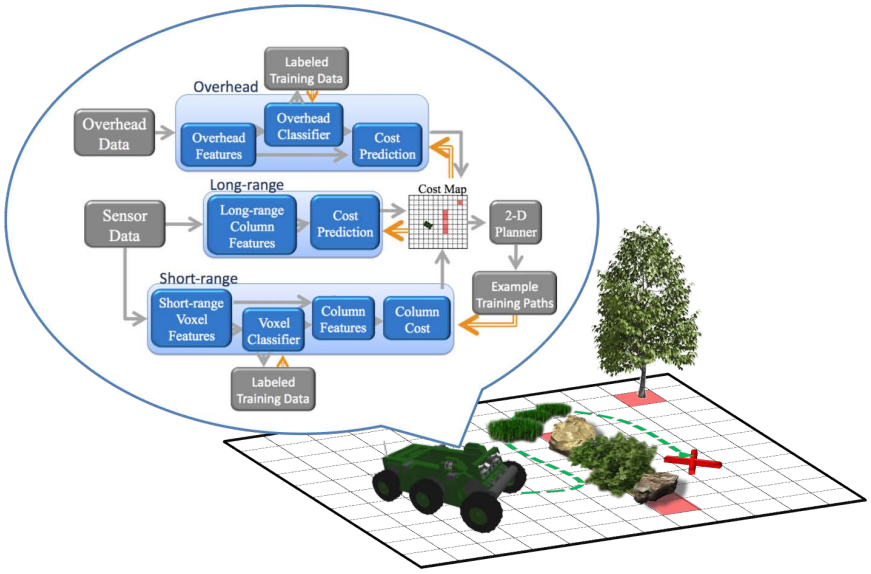


Figure 1.1: Shown is a state-of-the-art modular solution for the autonomous off-road navigation problem discussed in Section 3.1. The robot is tasked with predicting the "best" path (green line) to a goal location (red X) using satellite imagery and data from the onboard sensors.

One example of a robotics challenge that has both high-dimensional inputs and high-dimensional structured outputs is outdoor mobile robot navigation. In this task, a robot must drive autonomously from its current location to a goal location. The navigation problem can be divided into separate vehicle control and path planning problems. The planning problem, shown in Figure 1.1, consists of predicting a safe, efficient path from the robot's current position to an arbitrary goal location. The input for the planning problem consists of the current and goal locations of the robot, and high

dimensional data collected from satellite imagery as well as cameras and LADAR[1] sensors onboard the robot. The output, feasible paths, are highly structured, as paths must be continuous from the start location to the goal location. The key insight that has made the planning problem solvable is that it can be decomposed into a set of easier sub-problems. Each subproblem is solved by a separate *module* (as discussed in Section 3.1), and a system of modules is used to solve the *target* problem of navigation. Another common example are natural language processing tasks like machine translation, information extraction, and information retrieval. These are often solved by a pipeline approach where simpler sub-problems like part-of-speech tagging, and named recognition are performed first, and then provided as input for later processing (Figure 1.2).

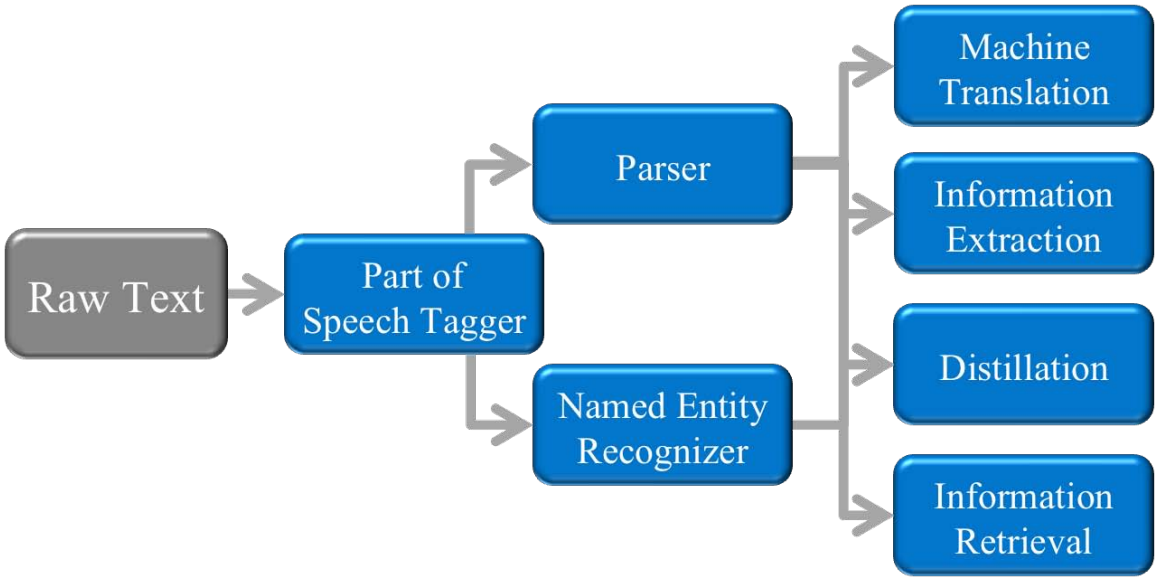


Figure 1.2: The pipeline approach common in natural language processing provides another example of a deep modular architecture.

In applications with high-dimensional inputs and/or structured outputs, it can be more efficient to use human intuition to design a modular system than as an oracle for labeling data. An effective modular design accomplishes several things: it creates subproblems which are well defined tasks, separates independent sources of variation in the overall task, and enables the solution of later subproblems. For instance, in the natural language processing example, the sub-problem of part-of-speech (noun, verb, etc...) prediction is easy to label data for, eliminates arbitrary sentence ordering variations for later follow-on processing, and makes other problems problems like named entity recognition easier to solve. In face recognition, it is often useful to have a separate face detection module which finds the location of probable faces in the image, and reduces the variation of the task by presenting the recognizer with an image of a standard size centered on the face.

Another way to think about learning is as a *noise reduction* problem. Modular systems can be designed to remove different types of noise in each module. In this view the input example is assumed to contain some information about the correct output, along with "noise" information

[1]LAser Detection And Ranging (LiDAR) sensors return 3-D range information.

that is unrelated to the task at hand. For instance, the image in Figure 1.3 shows a bush (front right) that the autonomous vehicle (center) can easily drive through, as well as a rock (front left) that must be avoided. Besides the local terrain, however, the image also shows the current weather conditions, the position of the sun, and the optical properties of the camera. If the target task is to classify the terrain in front of the robot, then the other information in the image is effectively noise that must be removed. Results from graphics research show that images can be effectively constructed in a modular way: a small set of parameters are sufficient to generate a wide array of realistic images, and recent results in computer vision suggest that fitting a similar set of modules to an image or image sequence is effective for extracting information of interest in the presence of noise information[8, 9, 10].

Figure 1.3: The autonomous robot "Crusher" (center), can drive through thick bushes (right) but must avoid rocks (left). Terrain classification from images is complicated by "noise" in the image from factors such as lighting weather, and camera optics.

Many of the modular designs for complex problems are *deep* in that they have many modules between the input and the output. This can increase the representational power dramatically, but it also creates new challenges for learning them. Results from circuit theory [11] show that for a given set of circuit elements, circuits of depth k can require exponentially fewer elements than circuits of depth $k-1$ to implement certain functions. Hence, deep architectures can theoretically represent complex functions with fewer parameters and lower computational requirements.

1.2 Information Theoretic Challenges

The representational power of deep modular systems presents a serious information-theoretic challenge for learning. Most of the prior work on learning in deep modular systems has focused on neural networks with several hidden layers. Until recently, attempts to train deep neural networks with more than two hidden layers largely failed in that they could learn the training set with near-perfect accuracy, without generalizing well to held-out test examples [12]. In other words, deep networks could minimize the *empirical risk* defined by the examples in the labeled training set without minimizing the *expected risk* of the test set. Learning theory establishes that learning is not likely to be successful unless the empirical risk provides a close approximation to the expected risk for the given hypothesis set, and hence minimization of the one will lead to minimization of the other. Training a deep neural network involves solving a non-convex optimization problem, and it is common knowledge that non-convex optimization procedures can get trapped in inferior local minima of of the empirical risk. However, if the size of each layer of the network is sufficiently large, the dominant problem is not optimization. It has been observed that when neural networks have large numbers of hidden units, they are unlikely to get trapped in a local minima [11]. It is likely that a large deep neural networks will have many stationary points and local minima that can achieve near-perfect accuracy on even a large training set. As the empirical risk is only an approximation of the expected risk, it is quite possible that the global minimum of the empirical risk will not be the global minimum of the expected risk. Hence, even if it were tractable to find a global minima of the training error, it would not necessarily help generalization performance.

To put this into information-theoretic terms, the representational power of wide, deep neural networks produces an enormous hypothesis class which is likely to overfit to even large training sets. Thus, a key challenge in deep learning is controlling the capacity of a deep network through regularization, so that optimization is likely to find a good local minima of the empirical risk (i.e. the layers of the network are sufficiently wide), and the local minima found by the optimization are likely to be close to *global* minima of the expected risk.

1.3 Data-dependent Regularization

It was recently shown [4, 3] that a pre-training step where the layers of the network are optimized to reconstruct their input, can lead to much better generalization performance after the network is trained to minimize empirical risk. The pre-training process replaces randomly-initialized parameters with parameters that have been tuned to maximize the likelihood of the input data. As will be discussed in Section 2.1 the effect of this pre-training is to add a *data-dependent regularization* function–or prior–on the parameters which is often better suited for the target task than a random or zero-centered prior would be because it is already adapted to model regularities in the input data used in the target task. For instance, if a deep network has been pre-trained to denoise handwritten digits, than a priori, before receiving any labeled training examples, the network is already set up to extract important structural information about digits and reject common noise, both of which are useful properties for digit recognition. More generally, if the information required for the target task is a significant part of the total energy of the input signal, then pre-training the network for

reconstruction will preserve information relevant to the target task, while eliminating other sources of noise that do not show regular structure in the data.

When the structure of a learning algorithm (i.e. the number of hidden units in a Neural Network) allows for a large hypothesis space, then the size of that hypothesis space must be controlled with regularization to avoid overfitting. Often regularization is implemented by a function penalizing the distance under a norm between each hypothesis and the zero vector, or a random zero-centered vector. The core idea behind data-dependent regularization is that the zero vector may be a poor choice for many problems, and performance can be improved by using *auxiliary* tasks to define a regularization function that is appropriate for the target function. The input reconstruction auxiliary task used in unsupervised pre-training is only one option, and a wide variety of other tasks have been used successfully in [13, 14, 15].

1.4 Optimization Challenges

It is easy to overlook the information theoretic challenges of deep modular systems when dealing with the significant optimization challenges they can present. In particular, it can be hard to determine how to assign credit (or blame) for the overall performance of the system to the parameters of individual modules in the system, a key component of any optimization process. In neural network training, credit is assigned by back-propagating gradients from a differentiable *loss* function that measures how the network's outputs differ from the desired behavior [1, 16]. When all of the elements of each layer are differentiable, this simply involves applying a chain rule.

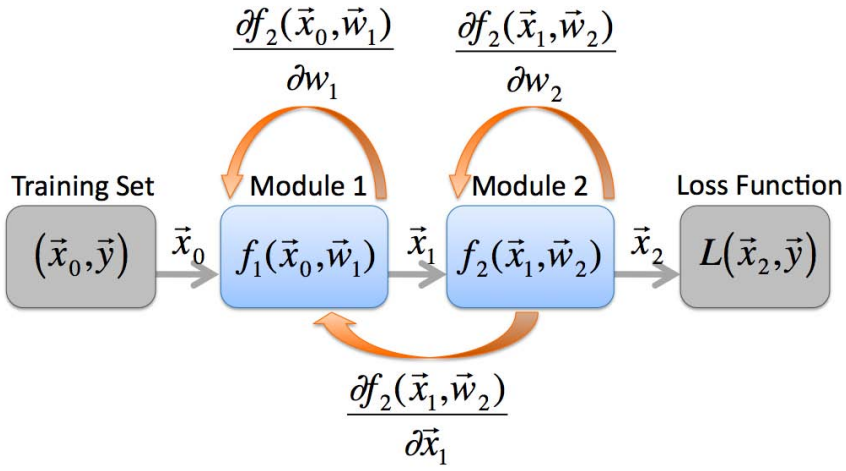


Figure 1.4: One way of training a modular system is by back-propagating error gradients. Each of the modules (blue boxes) is defined by a forward equation indicated by $x_i = f_i(\vec{x_{i-1}}, \vec{w_i})$. The parameter vector $\vec{w_i}$ of each module can be updated by computing the Jacobian of the modules outputs with respect to it's parameters (indicated by the top orange arrows). Loss gradient information is propagated to previous modules in the network by computing the Jacobian of the module outputs with respect to its inputs (bottom orange arrow).

As shown in Figure 1.4, the approach introduced by [1] assumes three equations for each module: a forward equation that computes the modules outputs given its inputs, a backward equation that computes the Jacobian $\frac{\partial f(\vec{x})}{\partial \vec{x}}$ of the outputs with respect to the inputs, and a gradient equation that computes the Jacobian $\frac{\partial f(\vec{x})}{\partial \vec{w}}$ of the outputs with respect to the parameters of the module. With these three equations for every module, gradients from the output loss function can be translated into gradients on any parameter in the system, and gradient descent can be used to jointly optimize the modules of the network.

In many robotics problems, credit assignment is complicated by inadequate representations, non-differentiable modules, and cascading errors. Standard gradient backpropagation will fail when the output of a module in the system provides an inadequate representation of the modules's input data, i.e. a representation which eliminates crucial elements of the target signal. For gradient backpropagation to work, there must initially be one or more outputs that are close to containing relevant information about the target signal. Those outputs will receive large loss gradients, and their parameters can be adjusted by multiplying the output gradients with the Jacobian of the module. However, if no outputs are close to providing relevant information about the prediction errors, then no outputs will receive significant loss gradients, and optimization can become trapped in an local minima or stationary point.

Non-differentiable modules add an obvious challenge to backpropagation; however, by taking a boosting approach [17, 18, 19, 20] it is possible to perform *functional* gradient descent on individual modules. This is a column-generation style approach where the complexity of individual modules is expanded as necessary, and it allows the use of a greater class of learning algorithms than standard backpropagation. Complicating these other challenges is that fact that small error in early modules can cascade into catastrophic errors in the output.

The sum of these challenges is that it can be hard to obtain information *local* to each module that defines how the parameters of the module can be changed to minimize the overall system error. Often it is hard even to define what task a intermediate module should be trying to solve. For instance, in the mobile robot navigation problem it is easy to believe that a module that classifies 3-D regions of space (voxels) into several terrain types would be useful as input for predicting the cost (negative reward) of driving over a particular piece of terrain. It is much harder, however, to define what those terrain types should be, and where the transition between classes should occur. Figure 1.5 shows an image taken by a mobile robot of a tree branch in the robot's path. At location A in the image, the branch may be placed in an "obstacle" category that the robot should avoid. At location B there is little risk of the branch damaging the robot, and it is better classified as dense vegetation which the robot can drive through if necessary. It is unclear when the class transition took place between A and B, as the performance of this voxel classification module is ultimately determined solely by its effect on the performance of the overall system on test data. Defining a training set of labeled examples of each terrain class can be helpful local information for the voxel classifier module, but the class definitions in the training set must be carefully constructed to allow the full system to perform well.

Figure 1.5: An image taken by a mobile robot shows an overhanging branch in the robot's path. At location A the branch is best classified as an "obstacle" class that the robot should avoid. At location B however, the branch is more similar to dense vegetation that the robot can drive through. Defining where the transition occurred presents a serious labeling challenge.

1.5 Adding Local Information

This thesis argues that overcoming regularization and credit assignment challenges when learning modular systems requires adding local information at each module, to guide how the parameters of that module are initialized and set, through data-dependent regularization, and generalized versions of backpropagation. The local information problem is particularly acute for modules far from the final output of the system. For these modules data-dependent regularization, through input reconstruction or prediction of auxiliary tasks, can be an especially powerful tool for solving the information theoretic challenges of deep systems. As mentioned earlier, input reconstruction has proven to be a useful source of local information for deep neural networks[4, 3], where it was used with Restricted Boltzmann Machines (RBM) and stacked auto-associators to initialize the parameters of each layer in the network. Sparse coding has also been shown to be a good way to learn from an input reconstruction task [21], but previous sparse coding modules could not be improved for a specific task by gradient descent, and that hampered their usefulness in prediction problems. Chapter 5, presents a novel method for *differentiable* sparse coding that allows a sparse coding module to be improved for a particular prediction task by backpropagation.

For some modules, other auxiliary tasks might be better related to the target task than input reconstruction, and consequently more useful as sources of local information. Chapter 3 outlines contributions that I have made to perception for mobile robot navigation, particularly in the development of a module that used a voxel classification auxiliary task to provide local information at

a key point in the overall perception system. A training set of labeled input output pairs, like the one used for the voxel classification task, defines an approximation joint probability distribution between labels and examples. Standard practice is to train the module to minimize regularized prediction error over the entire training set, however this makes the assumption that the robot is equally likely to observe examples from any of the areas of the country, seasons, and lighting conditions the labeled training set was drawn from. In many robotics applications, the distribution of examples that the robot will see changes slowly and as shown in Chapter 4 it can be advantageous to adapt to the current environment by sampling similar examples from the training set–effectively using only the conditional distribution of the data.

Often it is possible to define multiple auxiliary tasks for a single module, and Chapter 6 makes theoretical contributions by extending the mirror descent (potential-based learning) framework with a novel multi-task online learning algorithm. The theory developed for multi-task learning is also useful for the sparse coding problem. Training a sparse coding module is generally accomplished by a non-convex alternating optimization procedure which has many stationary points and local minima. In Chapter 7 I use the results of Chapter 6 to introduce a sparse coding algorithm that is convex over finite basis sets, and convex up to an oracle on infinite basis sets.

Chapter 8 examines ways to make information from errors on the target task local to the parameters of each module in the system. Modules which are close to the system output can often get sufficient local information from backpropagation alone, if the output module of the system is differentiable. However, non-differentiable output modules, such as the planning module used in the mobile robot navigation example, are crucial components of the system and cannot be replaced with differentiable modules. For these non-differentiable modules, subgradient backpropagation, and functional backpropagation (boosting) can be used to create local information for previous modules in the system. Another contribution of this chapter is the development of a new objective function for Maximum Margin Planning with exponentiated cost functions that corrects key undesirable behavior exhibited by the objective function function used in previous work [22].

1.6 Contributions

Modular systems are common in robotics, and a key challenge in learning modular systems, particularly deep ones, is obtaining local information about how to change the parameters of each module. This local information can come in two ways. First, regularization can be added to the parameters of a module, thereby adding prior information about "good" parameter values. The main contributions of this thesis are novel ways to add data-dependent regularization through sparse coding, multi-task learning, and domain adaptation. Secondly, past work has shown [3] that jointly optimizing the modules of a deep system for the target prediction task can significantly improve performance. However, many robotics applications use non-differentiable planning modules to create structured outputs, and a contribution of this thesis is developing techniques for subgradient backpropagation through planning modules. Finally, this thesis has also made significant contributions to mobile robot perception for navigation, providing terrain classification techniques that been used in fielded industrial and government systems [7].

Mobile Robot Perception

Chapter 3 discusses the UPI perception system for mobile robot navigation, and specific contributions made to the development of a terrain-classification module that uses local information from a terrain classification auxiliary task to enable state-of-the-art mobile robot navigation results in results in extensive field testing. Many of these research contributions are now part of fielded systems.

1. Range-image paradigm enabled long-range perception, reduced susceptibility to pose errors, and improved parallelism by allowing independent processing of each sensor (Section 3.2).

2. Advanced the state-of-the-art in "terrain classification", and particularly "vegetation detection", a critical problem in off-road navigation, by employing multispectral methods (Section 3.3).

3. Demonstrated how importance sampling can be used with unlabeled data samples from the robots current environment to automatically adapt the voxel classifier to the current operating environment (Chapter 4).

Advances in Imitation Learning

Maximum Margin Planning (MMP) [5, 23, 24, 22, 25, 26] has become a key tool for mobile robot navigation and other learning systems that use planning modules to produce structured outputs, and previous work has shown its effectiveness in learning shallow systems for a wide variety of applications. This thesis adapts MMP for use in deeper modular systems by:

1. Demonstrating the coordination of a system of modules by employing subgradient backpropagation to provide local information to earlier modules (Chapter 8).

2. Devising a modified MMP objective function that can be reliably optimized with gradient descent (Chapter 8).

Learning Theory

1. Derived the dual of general compositional norms (Section 6.2.1).

2. Constructed an online multi-task learning algorithm using compositional norms (Section 6.3).

3. Novel interpretation of boosting as Fenchel sub-gradient descent (Section 7.2.2).

4. Improved regret bound for the exponentiated gradient descent online-learning algorithm (Section 2.1.5).

Sparse Coding

1. Showed how implicit differentiation could be used to create differentiable sparse coding modules (Section 5.2.3).

2. Introduced KL regularization for sparse coding and demonstrated superior performance to the standard L_1 regularization (Section 5.2.2).

3. Convex (up to an oracle) formulation of the sparse coding problem using $L_{2,1} + \gamma L_1$ prior which relaxes the non-convex rank constraint implicit in traditional sparse coding by using a convex compositional (block) norm (Section 7.1).

4. Proposed an efficient heuristic for solving the oracle sub-problem in convex sparse coding (Section 7.3).

1.7 Notation

Uppercase letters, X, denote matrices and lowercase letters, x, denote vectors. For matrices, superscripts and subscripts denote rows and columns respectively. X_j is the j^{th} column of X, X^i is the i^{th} row of X, and X^i_j is the element in the i^{th} row and j^{th} column. Elements of vectors are indicated by subscripts, x_j, and superscripts on vectors are used for time indexing x^t. X^T is the transpose of matrix X.

Chapter 2

Deep and Online Learning

This chapter contains background information on the applications and the theoretical results in online and deep learning that inspired this thesis. Section 2.1 provides background from the field of online learning helpful for understanding the sparse coding and multi-task learning results presented in Chapters 6 & 7. The online learning background is also useful for Section 2.2, which describes recent results in deep learning and how they relate to the methods examined in this thesis.

2.1 Lessons from Online Learning

This section introduces the online learning framework, and key concepts such as *regret* and *norms*. The competitive regret bounds of online learning algorithms, particularly under the potential-based gradient descent framework, are closely related to boosting. To summarize, the potential-based gradient descent steps offer a controlled update step that resists moving fast enough to be tricked, while still being fast enough to reach any competing expert in a norm-ball quickly (i.e. compete with any fixed expert).

2.1.1 Online Learning Framework

The convex online learning framework [27] consists of a series of rounds where on round t the learning algorithm is asked to select a hypothesis $\hat{w}_{t-1} \in \mathfrak{W}$, where $\mathfrak{W}$ is the *hypothesis space*, or set of available hypotheses. For example, in linear classification each hypothesis is a vector of weights. The algorithm is then presented with an example x_t, y_t generated by an adversary, where $x_t \in X$ is a vector from an m-dimensional input space, and $y_t \in Y$ is an element from the output space. The algorithm then pays a penalty according to a convex loss function $L_t(f(x_t^T \hat{w}_{t-1}), y_t)$ between the given output for the example y_t, and output of a transfer function $f(r)$ which is a function of the dot product between the input vector and the hypothesis $r = x_t^T \hat{w}_{t-1}$. The *regret* $R_t(w^*)$ the learning algorithm suffers on the first t rounds is the difference between the loss suffered by the sequence of hypotheses $\hat{w}_1, ..., \hat{w}_t$ chosen by the learning algorithm, and the loss suffered by the fixed[1] hypothesis w^* (2.1). In particular, w^* is often assumed to be the lowest-loss fixed hypothesis, in order to bound

[1] Regret is a general idea, and has also been expanded to other classes of hypotheses.

the maximum size of the regret. Note that under this framework the loss function can be a *different* convex function for each round t, so for simplicity of notation it will be referred to simply as $L_t(w)$.

$$R_t(w^*) = \sum_{\tau=1}^{t} L_\tau(\hat{w}_{\tau-1}) - L_\tau(w^*) \tag{2.1}$$

An online learning algorithm must select a sequence of hypotheses $\hat{w}_0, ..., \hat{w}_{t-1}$ that suffers low regret, while being efficient enough to operate indefinitely with fixed computational and memory resources. At time $t + 1$ the best fixed hypothesis w^* satisfies:

$$w^* = \arg\min_{w \in \mathfrak{W}} \sum_{\tau=1}^{t+1} L_\tau(w)$$

Of course the learning algorithm does not have access to the loss function for round $t + 1$ before it selects a hypothesis to use. Instead the hypothesis used, denoted $\hat{w}_t$ because it has only been informed by the first t examples, optimizes the regret on the previous rounds plus a regularization function (or prior function), $\Phi(w)$:

$$\hat{w}_t = \arg\min_{w \in \mathfrak{W}} R_t(w) + \lambda\Phi(w). \tag{2.2}$$

Maximum likelihood (unregularized) learning can suffer high regret

The regularization function $\Phi(w)$ ensures that the solution is not too far from w^*, and without it unlucky or adversarial sequences of inputs can cause $\hat{w}_t$ to change rapidly and suffer high regret in the online case, or overfitting in the offline case. For example, consider the simple case where $L_t(w) = |w - y| \; \forall t$, and y follows the sequence $\{1, -1, -1, 1, 1, -1, -1, 1, \dots\}$. In this case, over time all fixed $w \in [-1, 1]$ will do equally well in the long run, with an average loss per round of 1. However, directly minimizing the loss over the previous examples produces the sequence $\{w_0, 1, w_0, -1, w_0, \dots\}$, where w_0 is the initial guess, which has an average loss of 1.5. The regret of the unregularized algorithm will thus grow linearly with the number of rounds, and the average regret will never converge to the average regret of the best fixed expert. The role of $\Phi(w)$ is to keep the minimum of (2.2) from changing too quickly, thereby limiting the possible regret with respect to any fixed vector.

2.1.2 Mirror Descent

Mirror descent (or potential-based gradient descent [28]) algorithms can accomplish online learning with guaranteed low regret because they minimize an objective function consisting of a bound on the regret plus a regularization term $\Phi(w)$ that places a prior on the weight vector w. The minimum of this objective function can be viewed as the Maximum A Posteriori (MAP) estimate $\hat{w}$. By definition, at $\hat{w}$ a subgradient of the prior will be equal to a subgradient of the regret. If the prior is a *Legendre function* with *Legendre dual* $\Phi^*(r)$, the MAP estimate $\hat{w}$ can be computed in closed form as the derivative of the Legendre dual applied to the gradient of the regret. For these priors, mirror descent also provides a way to bound the regret in terms of the *Bregman divergence* defined by the

prior. I will show in Chapter 7 that the mirror descent idea can also be extended to non-differentiable prior functions, where it allows the creation of boosting algorithms for a large class of priors.

Approximating Convex Loss Functions

If the loss functions L_t are assumed to be convex, we can simplify the regularized loss minimization in (2.2) by upper bounding the regret[2] with the linear approximation to each convex loss function given by the first-order Taylor expansion (2.3):

$$\begin{aligned} L_\tau(w) &\geq L_\tau(\hat{w}^{\tau-1}) + (w - \hat{w}^{\tau-1})^T \nabla L_\tau(\hat{w}^{\tau-1}) \\ R_t &\leq \sum_{\tau=1}^{t} L_\tau(\hat{w}^{\tau-1}) - L_\tau(\hat{w}^{\tau-1}) + (w - \hat{w}^{\tau-1})^T \nabla L_\tau(\hat{w}^{\tau-1}). \\ R_t &\leq \sum_{\tau=1}^{t} (w - \hat{w}^{\tau-1})^T \nabla L_\tau(\hat{w}^{\tau-1}) \end{aligned} \tag{2.3}$$

In many cases the complete loss function for each round is not available anyway, and the results obtained with the linear bound lead to an efficient algorithm and hold true for any series of arbitrary convex loss functions. Substituting the regret bound (2.3) into (2.2) produces a more tractable optimization problem:

$$\hat{w}_{t-1} = \arg\min_{w \in \mathfrak{W}} \sum_{\tau=1}^{t-1} (w - \hat{w}^{\tau-1})^T \nabla L_\tau(\hat{w}^{\tau-1}) + \lambda \Phi(w) \tag{2.4}$$

Efficient MAP updates with Legendre Duality

Legendre functions are useful for efficiently solving (2.4). Following the convention of [28] A function $f : \mathfrak{W} \to \Re$, is Legendre if:

1. $\mathfrak{W} \subset \Re^d$ is non-empty and its interior int($\mathfrak{W}$) is convex.
2. f is strictly convex with continuous first partial derivatives throughout int($\mathfrak{W}$).
3. if $w_1, w_2, ..., \in \mathfrak{W}$ is a sequence converging to a boundary point of $\mathfrak{W}$, then $\|\nabla f(w_n)\| \to \infty$ as $n \to \infty$.

The Legendre dual f^* of f is commonly defined as (2.5), which has the important property that the gradients of f and its dual f^* are inverse functions $\nabla f = (\nabla f^*)^{-1}$.

$$f^*(u) = \sup_{v \in \mathfrak{W}} u \cdot v - f(v) \tag{2.5}$$

In the general case when f is not assumed to be differentiable, (2.5) is known as the *Conjugate* or *Fenchel Conjugate* function [29]. If Φ is differentiable, then at $\hat{w}_t$, the gradient of the regularization

[2] For clarity this discussion uses the regret with respect to the minimum loss hypothesis w^*, but the same equations hold true for any arbitrary vector $u \in \mathfrak{W}$.

function will equal the negative gradient of the regret, and we substitute the regret upper bound (2.3) to produce:

$$\lambda \nabla \Phi(\hat{w}_t) = -\sum_{\tau=1}^{t} \nabla L_\tau(\hat{w}^{\tau-1}). \tag{2.6}$$

Furthermore, if the gradient of the regularization function is invertible, $\hat{w}_t$ can be computed directly from the regret gradient (2.6). In this case the integral of $\nabla\Phi^{-1}$, Φ^*, is known as the Legendre dual of the prior $\Phi(w)$, and the space of regret gradients, is the "mirror" space to the space of weight vectors. $\nabla\Phi^*$ allows simple additive gradient updates in the "mirror" space to be converted easily into MAP estimates $\hat{w}_t$:

$$\hat{w}_t \quad = \quad \nabla\Phi^{-1}\left(-\frac{1}{\lambda}\sum_{\tau=1}^{t} \nabla L_\tau(\hat{w}^{\tau-1})\right). \tag{2.7}$$

Example: Stochastic Gradient Descent (SGD)

The update algorithm used with backpropagation, Stochastic Gradient Descent (SGD), is derived from applying mirror descent with $\Phi(w) = \|w\|_2^2 = \sum_i w_i^2$. Substituting this prior into (2.7) produces (2.8), where $\hat{w}^t$ is simply a scaled version of the sum of the gradients for the first t rounds. The scale factor, $1/\lambda$, serves as the step size in the recursive version of the algorithm (2.9).

$$\hat{w}^t \quad = \quad -\frac{1}{\lambda}\sum_{\tau=1}^{t} \nabla_w L_\tau(\hat{w}^{\tau-1}) \tag{2.8}$$

$$\hat{w}^{t+1} \quad = \quad \hat{w}^t - \frac{1}{\lambda}\nabla_w L_{t+1}(\hat{w}^t) \tag{2.9}$$

Using Bregman divergences, it has been shown that the regret suffered by SGD is bounded by (2.10), for the step size $1/\lambda = \frac{G_2\|w^*\|_2}{\sqrt{t}}$. Here G_2 is a constant that bounds the L_2 norm of the gradient of the instantaneous regret r_t for any round.

$$R_t \leq \|w^*\|_2 G_2 \sqrt{t} \tag{2.10}$$

Bregman Divergences

Every Legendre function induces a Bregman divergence $D_f(p\|q)$, which is the difference between $f(p)$ and the first order Taylor series expansion of f around q. Since f is strictly convex, this function is non-negative for any two distinct points $p, q \in \mathfrak{W}$ (2.11).

$$D_f(p\|q) = f(p) - f(q) - (p-q)^T \nabla f(q) \tag{2.11}$$

Although a Bregman divergence is not a metric[3], it has several useful distance-like properties. One that will be especially useful here is the relationship between the Bregman divergences of any three arbitrary points u, v, and w.

$$D_f(u\|v) + D_f(v\|w) - D_f(u\|w) = (v-u)^T(\nabla f(v) - \nabla f(w)) \tag{2.12}$$

[3]Bregman divergences do not satisfy the triangle inequality or symmetry

There is also an important relationship between the Bregman divergences of dual functions. If $u' = \nabla f(u)$ and $v' = \nabla f(v)$, then the divergence under f from u to v is equal to the divergence under f^* from v' to u' (2.13).

$$D_f(u\|v) = D_{f^*}(v'\|u') \tag{2.13}$$

2.1.3 Bounding the regret of an online algorithm

Differentiating the upper bound on the regret (2.3) with respect to w, we see that the sum of the regret gradients at the best fixed weight vector is equal to the sum of the loss function gradients. Since the instantaneous regret, $r_t = R_t - R_{t-1}$, we can use (2.6) to get an upper bound on the instantaneous regret in terms of the difference in prior gradients between $\hat{w}_{t-1}$ and $\hat{w}_t$:

$$\begin{aligned} \nabla_w R_t &\leq \sum_{\tau=1}^{t} \nabla L_\tau(\hat{w}^{\tau-1}) \\ \nabla r_t &\leq \nabla L_t(\hat{w}^{t-1}) = \lambda\left(\nabla\Phi(\hat{w}^{t-1}) - \nabla\Phi(\hat{w}^t)\right). \end{aligned} \tag{2.14}$$

If the prior Φ is chosen to be a Legendre function, (2.14) can be used to bound the regret in terms of the bregman divergences defined by Φ and its Legendre dual Φ^*.

Substituting the relationship between the gradient of the prior and the gradient of the loss (2.14) into the regret bound (2.3) produces a bound that depends only on points in the hypothesis space and Φ, which is a function over the hypothesis space:

$$R_t \leq \lambda \sum_{\tau=1}^{t} (w - \hat{w}^{\tau-1})^T \left(\nabla\Phi(w^{\tau-1}) - \nabla\Phi(w^\tau)\right). \tag{2.15}$$

If Φ is Legendre, the relationship between the Bregman divergences of any three arbitrary points (2.12) can be used to rewrite this regret bound in terms of D_Φ, the Bregman divergence associated with Φ:

$$R_t \leq \lambda \sum_{\tau=1}^{t} D_\Phi(w^*\|w^{\tau-1}) - D_\Phi(w^*\|w^\tau) + D_\Phi(w^{\tau-1}\|w^\tau). \tag{2.16}$$

The sum in (2.16) telescopes and the final term, $D_\Phi(w^*\|w^t)$, is a Bregman divergence and thus non-negative. What remains in the bound is the distance under the prior between w_0 and w^*, plus a term that measures how large of a step the solution took at each time-step:

$$\begin{aligned} R_t &\leq \lambda\left(D_\Phi(w^*\|w^0) - D_\Phi(w^*\|w^t) + \sum_{\tau=1}^{t} D_\Phi(w^{\tau-1}\|w^\tau)\right) \\ R_t &\leq \lambda D_\Phi(w^*\|w^0) + \lambda \sum_{\tau=1}^{t} D_\Phi(w^{\tau-1}\|w^\tau). \end{aligned} \tag{2.17}$$

Note that decreasing λ will cause the algorithm to take bigger steps, and $D_\Phi(w^{\tau-1}\|w^\tau)$ will increase. Hence, it is not possible to minimize (2.17) simply by setting $\lambda = 0$ (the maximum likelihood

case). Minimizing this bound requires a prior which keeps the path the algorithm takes through the hypothesis space short and straight, i.e. the distance between the prior w_0 and the vector w^* small, and the total distance traveled (the sum of the size of each step) is small. Often the total distance traveled is easier to measure using the dual of Φ, and (2.13) can be used to rewrite the second term of the bound in terms of the dual variable $\theta^t = \sum_{\tau=1}^{t} -\frac{1}{\lambda}\nabla L_\tau(w^{\tau-1})$ (2.18).

$$R_t \leq \lambda D_\Phi(w^*||w^0) + \lambda \sum_{\tau=1}^{t} D_{\Phi^*}(\theta^\tau||\theta^{\tau-1}) \tag{2.18}$$

Derivation of the stochastic gradient descent regret bound

Essentially, SGD is the intuitive idea of taking a small step after each example in the direction that would have most reduced the loss suffered on that example, i.e. the negative gradient. In contrast to batch gradient descent, which uses the gradient of the loss over all examples for each step, SGD takes a smaller step after each example, using the loss gradient on that example as a noisy estimate of the total sample gradient. This approach allows online learning and speeds up initial convergence, especially when the training data contains many similar examples.

Analyzing SGD using (2.18), is particularly straightforward since the squared L_2-norm is its own dual, $\Phi^*(\theta) = \frac{1}{2}||\theta||_2^2$, and the associated Bregman divergence from u to v is just the sum of squared differences $D_\Phi(u||v) = \frac{1}{2}||u - v||_2^2$ (2.19). Hence $D_{\Phi^*}(\theta^\tau||\theta^{\tau-1}) = ||\frac{1}{\lambda}\nabla L_\tau(w^{\tau-1})||_2^2$, and the regret bound becomes:

$$R_t \leq \lambda\frac{1}{2}||w^* - w^0||_2^2 + \frac{1}{\lambda}\sum_{\tau=1}^{t}||\nabla L_\tau(w^{\tau-1})||_2^2 \tag{2.19}$$

If the magnitude of the loss gradient, $||\nabla L_\tau(w^{\tau-1})||_2$, can be bounded by a constant G_2, this gives a simple, easy to interpret bound:

$$R_t \leq ||w^*||_2 G_2\sqrt{t}, \quad \frac{1}{\lambda} = \frac{||w^*||_2}{G_2\sqrt{t}}. \tag{2.20}$$

Regret Bounded in Terms of Holder-conjugate Norms

An important thing to notice in (2.20) is that the regret of the algorithm depends on the L_2 norms of the loss gradients (which is generally closely related to the L_2 norm of the input) and the optimal weight vector w^*. In general, mirror descent bounds trade off these quantities using a Holder inequality where the $L_p,\ p \geq 2$ norm used on the gradient space is the holder conjugate[4] of the $L_q,\ 1 < q \leq 2$ norm used on the weight vectors. The choice of norms is important as the magnitude of an L_p norm *monotonically* decreases with increasing p, $||X||_p \geq ||X||_{p+\epsilon}\ \forall\epsilon > 0$, and the size of the decrease can be quite significant. For instance a vector $\vec{1}$ of dimension d containing all ones has $||\vec{1}||_1 = d$, $||\vec{1}||_2 = \sqrt{d}$, and $||\vec{1}||_\infty = 1$. Thus, if the optimal weight vector w^* is expected to have many large values and the input vectors are normalized to a small L_2 magnitude or are sparse (i.e. expected to have only a few elements that are significantly larger than zero), then we should choose

[4]Holder p and q are holder conjugates if $1/p + 1/q = 1$.

the SGD algorithm as its regret bound will depend on $\|w^*\|_2 G_2$ which will be less than any other holder norm pair $\|w^*\|_p G_q$, where $p < 2$.

However, it can be hard to select a low-dimensional or sparse input space that solves the learning problem well, and it would be much easier to throw in many possible features and let the classifier find a small subset of those features to use. In that approach the input would be high-dimensional and dense (i.e. most features are significantly non-zero in most examples), and the weight vector would have large weights on only a few of the input features. Then the bound (2.10) could be much improved by applying an L_∞ norm on the inputs and an L_1 norm on the weight vectors. The Exponentiated Gradient Descent (EGD) algorithm discussed next uses an implicit KL-divergence prior to provide such a bound. The connection between the L_1 norm and the KL-divergence explored in bounding the regret of the EGD algorithm will also be useful in Chapter 5, where it will enable a novel semi-supervised learning algorithm using a differentiable form of sparse coding.

2.1.4 Example: Exponential Gradient Descent

The unnormalized KL-divergence prior function $\Phi(w) = KL(w\|w^0) = \sum_i w_i \log \frac{w_i}{w_i^0} - w_i + w_i^0$, produces the Exponential Gradient Descent (EGD) algorithm [30]. The dual of the unnormalized KL-divergence is a weighted sum of exponentials $\Phi^*(\theta) = \sum_i w_i^0 e^{\theta_i}$, and applying (2.7) produces the update rule (2.21).

$$\begin{aligned} \hat{w}^{t+1} &= w^0 e^{-\frac{1}{\lambda}\sum_{\tau=1}^{t+1} \nabla_w L_\tau(\hat{w}^{\tau-1})} \\ &= \hat{w}^t e^{-\frac{1}{\lambda}\nabla_w L_{t+1}(\hat{w}^t)} \end{aligned} \tag{2.21}$$

A contribution of this thesis is a new, tighter bound on the regret of this algorithm (2.22). The proof of this bound is given in Section 2.1.5. The key features of (2.22) are a L_∞ norm on the loss gradient, and a logarithmic dependence on the number of dimensions d of the input. This makes EGD well suited for situations where the input vectors are dense and contain many irrelevant dimensions. If only a small number of input features have non-zero weights in the best solution, the fact that the L_1 norm on the weight vectors is larger than the L_2 norm used by SGD is outweighed by the gain in switching to the L_∞ norm on the loss gradient.

$$R_t \leq 2F_1 G_\infty \sqrt{t\alpha \log d} \tag{2.22}$$

Where $\alpha \geq \frac{\lambda}{G_\infty} + \frac{\lambda^2}{G_\infty^2}\left(e^{\frac{\lambda}{G_\infty}} - 1\right)$.

2.1.5 Bounding the regret of Exponentiated Gradient Descent

A contribution of this thesis is a new, tighter bound on the regret of the Exponentiated Gradient Descent (EGD) algorithm (Described in Section 2.1.4) which is obtained by using a second-order Taylor approximation on the dual of the implicit prior implemented by the EGD update rule. This prior, $\mathrm{KL}(\vec{w}\|\vec{p}) = \sum_i w_i \log \frac{w_i}{p_i} - w_i + p_i$, is dual to $\Phi^*(\vec{\theta}) = \sum_i p_i e^{\theta_i}$, where θ is again defined as the sum of the negative loss gradients weighted by the step size. The Bregman divergence of this dual,

$D_{\Phi^*}(\vec{a}\|\vec{b})$ can be written as:

$$D_{\Phi^*}(\vec{a}\|\vec{b}) = \sum_i p_i e^{b_i}(e^{a_i - b_i} - a_i + b_i - 1). \tag{2.23}$$

Substituting (2.23) into (2.18) produces (2.25), which uses the easily verified fact that the Bregman divergence under the prior $\Phi^*(\vec{a}) = \mathrm{KL}(\vec{a}\|\vec{b})$ between any two vectors $\vec{a}$ and $\vec{b}$ is simply $\mathrm{KL}(\vec{a}\|\vec{b})$.

$$R_t \leq \lambda \mathrm{KL}(\vec{w}^*\|\vec{p}) + \lambda \sum_{\tau=1}^{t} D_{\Phi^*}(\theta^\tau \| \theta^{\tau-1}) \tag{2.24}$$

$$\begin{aligned} R_t &\leq \lambda \mathrm{KL}(\vec{w}^*\|\vec{p}) + \lambda \sum_{\tau=1}^{t} \sum_i p_i e^{\theta_i^{\tau-1}}(e^{\theta_i^\tau - \theta_i^{\tau-1}} - \theta_i^\tau + \theta_i^{\tau-1} - 1) \\ &= \lambda \mathrm{KL}(\vec{w}^*\|\vec{p}) + \lambda \sum_{\tau=1}^{t} \sum_i w_i^{\tau-1}(e^{-\nabla_i L_\tau(w^{\tau-1})/\lambda} + \frac{\nabla_i L_\tau(w^{\tau-1})}{\lambda} - 1) \end{aligned} \tag{2.25}$$

If the prior vector $\vec{p}$ is uniform with dimension d, $\mathrm{KL}(\vec{w}^*\|\vec{p})$ can be rewritten as (2.26). Because log is a concave function, the definition of concavity can be applied to bound the sum of logs as a log of a sum $\|w\|_1 \sum_i \frac{w_i}{\|w\|_1} \log w_i \leq \|w\|_1 \log\left(\sum_i \frac{w_i^2}{\|w\|_1}\right)$. A general property of Holder norms is: $\|x\|_p \leq \|x\|_q, \forall x$ when $p \geq q$. Thus as long as the prior vector is larger than the best constant weight vector, $\|\vec{p}\|_1 \geq \|\vec{w}^*\|_2$, (2.26) can be bounded by (2.27). Assuming that $d \geq 3$, and that there is a constant F_1 such that $F_1 \geq \|\vec{p}\|_1 \geq \|\vec{w}^*\|_1$, then equation (2.28) holds.

$$\begin{aligned} \mathrm{KL}(\vec{w}^*\|\vec{p}) &= \sum_{i=1}^{d} w_i^* \log w_i^* + w_i^* \log \frac{d}{\|\vec{p}\|_1} - w_i^* + p_i && \forall i,\ p_i = \frac{\|\vec{p}\|_1}{d} \\ &\leq \|\vec{w}^*\|_1 \log \frac{\|\vec{w}^*\|_2^2}{\|\vec{w}^*\|_1} + \|\vec{w}^*\|_1 \log \frac{d}{\|\vec{p}\|_1} + \|\vec{p}\|_1 - \|\vec{w}^*\|_1 && \text{concavity} \\ &\leq \|\vec{w}^*\|_1 \left(\log d + \log \frac{\|\vec{w}^*\|_2^2}{\|\vec{w}^*\|_1 \|\vec{p}\|_1}\right) + (\|\vec{p}\|_1 - \|\vec{w}^*\|_1) && (2.26) \\ &\leq \|\vec{w}^*\|_1 \log d + (\|\vec{p}\|_1 - \|\vec{w}^*\|_1) && \|\vec{p}\|_1 \geq \|\vec{w}^*\|_2 \quad (2.27) \\ &\leq F_1 \log d && F_1 \geq \|\vec{w}^*\|_1 \geq \|\vec{p}\|_1 \quad (2.28) \end{aligned}$$

Now onto the second term of the bound. As shown in figure 2.1, e^{-x} can be upper-bounded by $1 - x + \alpha x^2$ when $\alpha \geq \frac{e^{-x}+x-1}{x^2}$. Applying this approximation to (2.25) produces the simpler bound (2.29).

$$R_t \leq \lambda F_1 \log d + \frac{\alpha}{\lambda} \sum_{\tau=1}^{t} \sum_i w_i^{\tau-1} \left(\nabla_i L_\tau(w^{\tau-1})\right)^2 \tag{2.29}$$

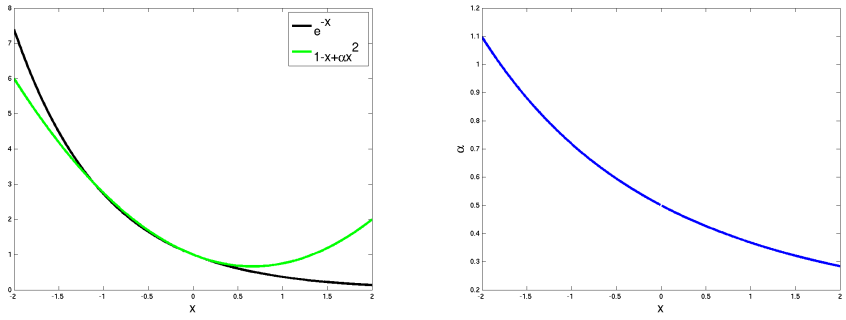


Figure 2.1: Left: e^{-x} is upper bounded by $1 - x + \alpha x^2$ for sufficiently large α (Shown is $\alpha = 0.75$). Right: The minimum value that x can take on determines the value of α needed to ensure the bound holds. As this approximation is used in the bound, x is the loss gradient divided by the step size.

The holder inequality can be applied to the second term of (2.29) to bound the inner sum in terms of a pair of Holder conjugate norms[5] $\|\vec{w}^{\tau-1}\|_u$ and $\|\left(\nabla L_\tau(w^{\tau-1})\right)\|_{2v}^2$ on the weight vector and the squared gradient[6]. If the norms can also be bounded by constants, $F_u \geq \|\vec{w}^{\tau-1}\|_u$, and $G_v \geq \|\left(\nabla L_\tau(w^{\tau-1})\right)^2\|_v$ for all τ, the bound becomes much simpler (2.30).

$$R_t \leq \lambda F_1 \log d + \frac{\alpha}{\lambda} \sum_{\tau=1}^{t} \|\vec{w}^{\tau-1}\|_u \|\left(\nabla L_\tau(w^{\tau-1})\right)\|_{2v}^2$$
$$R_t \leq \lambda F_1 \log d + \frac{\alpha t}{\lambda} F_u G_{2v}^2 \tag{2.30}$$

Since the first term is bounded in terms of L_1 norms on weight vectors, a natural choice for the holder conjugate norms in (2.30) is F_1 and G_∞. Substituting (2.28) in to (2.30) and choosing the regularization constant $\lambda = G_\infty \sqrt{\frac{\alpha t}{\log d}}$ produces the final bound (2.31). For this constant step size the average regret, $\bar{R}_t$ decays as $O(1/\sqrt{t})$ (2.32).

$$R_t \leq \lambda F_1 \log d + \frac{\alpha t}{\lambda} F_1 G_\infty^2$$
$$R_t \leq 2F_1 G_\infty \sqrt{t\alpha \log d} \tag{2.31}$$
$$\bar{R}_t \leq 2F_1 G_\infty \sqrt{\frac{\alpha \log d}{t}} \tag{2.32}$$

2.1.6 Lessons from Online Learning

A key insight from online learning that is important for deep modular systems is that taking a series of small update steps and stopping before convergence, so called *early stopping*, is a form of regularization around the initial parameters of the system. For example, the stochastic gradient descent

[5] $\frac{1}{u} + \frac{1}{v} = 1$
[6] $\|x^2\|_q = \|x\|_{2q}^2$

algorithm implicitly uses an L_2 regularization function, whose importance is inversely proportional to the step size. Therefore, optimizing a module's parameters for one or more auxiliary tasks that are related to the target task before training on the target task with SGD and early-stopping is a form of data-dependent regularization.

2.2 Deep Learning

The *depth* of a modular system is the number of non-linear modules on the longest path from the input to the output. Traditionally, shallow architectures with only one or two non-linearities have been most useful in machine learning. For complicated, highly varying functions, however, deep architectures are necessary to learn with reasonable sample complexity. The limitations of shallow architectures discussed below mean that in practical applications the problem setup–the definition of the feature space that the learner uses as input and the output required of the learner– is the most critical component of developing a successful machine learning solution. Current practice leaves the modules that define the feature space and the modules that post-process the output out of the automatic training process. This imposes an awkward hybrid design process where human intuition is used to improve the input and output modules, as well as the labeled training set, by guessing what effect changes will have on the automatic data-driven learning process. The goal of deep modular learning is to understand how the modules of a deep system can be regularized and jointly optimized to achieve performance gains and reduced engineering expense. After reviewing the limitations of shallow architectures, this section will discuss the information theoretic challenges of deep learning and how data-dependent regularization methods have been key to current successes in machine learning.

2.2.1 Limitations of Shallow Architectures

An insightful analysis of the limitations of shallow architectures is provided in [11]. To summarize: a wide variety of shallow architectures are based on one level of local template matching, where a distance is computed between the input and a set of prototype examples. The output is then predicted by interpolating the prototypes close to the input, generally with a simple function like a linear combination. Many standard machine learning algorithms such as Kernel Machines (such as Support Vector Machines (SVMs) and Gaussian Processes), 2-layer Neural Networks, k Nearest Neighbors (kNN), decision trees, and regression trees fit into this paradigm. When the target task has a small number of variations in the input space, i.e. a piecewise linear approximation of the function is accurate with a small number of pieces, then template matching can learn with a reasonable number of examples. However, local template matching approaches are insufficient for *complicated* functions that have many more variations than training examples, as in that case there is no reasonable expectation that there will be examples close to most of the variations.

The consequence of this line of analysis is that to learn from a limited number of training examples, shallow architectures require either an input feature space where the target function is smooth, or a specially designed kernel that implicitly defines a feature space where the target is smooth. In many applications the raw input data is a poor feature space where the target function is not smooth. Hence a critical part of the success of these applications is defining functions that

transform the raw input into a good feature space (or equivalently engineering kernel functions).

Even if the number of labeled training examples were not an issue, shallow architectures can be infeasible because of the computational cost of comparing to the templates, and the memory required to store them. Results from circuit theory [11] suggest that deep architectures provide a solution to this problem by allowing compact representations of complicated functions. Some functions that can be represented compactly with an architecture of depth k require exponentially more circuit elements to be expressed with depth k-1. Hence for some functions deep architectures can provide a solution to the computational problem, but in practice deep architectures can be particularly hard to learn.

Do the challenges in learning with deep architectures arise from information theoretic or optimization limitations? The information theoretic PAC-learning approach [31] examines the total number of training examples (i.e. sample complexity) required to select a hypothesis from the total set of hypotheses available to the algorithm that is Probably, Approximately Correct, i.e. with high probability the selected hypothesis will have low generalization error. PAC learning does not consider the computational cost of selecting a hypothesis, only whether, given infinite computation, such a hypothesis can be reliably selected. The sample complexity of PAC learning for a given class of functions has been shown to be linear in the VC dimension of the function [32], for identical training and testing distributions. In [33, 34] it was shown that the number of training samples required for PAC learning of a neural network is bounded by a function of the total number of weights w in the network, and is at most $O(w\log(w))$ independent of the depth of the network. Maass, [35], showed that this bound is tight for a class of networks of depth greater than two.[7] Hence from a PAC-learning perspective, the information theoretic difficulty of learning an *arbitrary* deep network is not much different from the problem of learning an arbitrary shallow network with the same number of weights. However, in practice we are not interested in learning arbitrary networks from a sufficient number of samples, we are interested in learning networks that solve real-world problems, typically with only an insufficient number of examples. The distinction is that choosing a network architecture and a regularization strategy appropriate to the problem at hand is critical. Recent advances have shown that new regularization methods for deep networks can make them more appropriate for several real-world applications than shallow networks.

2.2.2 Information Theoretic Challenges

Pluralitas non est ponenda sine necessitate.
Entities are not to be multiplied beyond necessity.
–William of Ockham

Regularization

Mathematically, "learning" is the problem of selecting between a set of hypotheses represented as parameter vectors. Learning algorithms define an objective function over a set of hypotheses that scores each hypothesis, and an optimization strategy for searching in the hypothesis space. The goal

[7] Although the VC-dimension is super-linear in the number of weights, the VC-dimension of Maass's class of networks was linear in the total number of bits used to represent the network. Hence one of these neural networks with N weight bits will have the same asymptotic sample complexity as selecting from a discrete set of 2^N arbitrary functions.

of learning is to find a hypothesis $\hat{\theta}$ that minimizes the expected loss (number of mistakes) on future examples:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \int_x L(\theta, x) p(x) dx. \tag{2.33}$$

Probabilistically, $L(\theta, x)$ in (2.33) can be interpreted as the log of a likelihood function $P(x|\theta) = \frac{1}{\mathcal{Z}} e^{-\eta L(\theta, x)}$, that expresses the probability of seeing the input vector x given that the parameters of the model are θ. $\mathcal{Z}$ and η are constants that ensure $P(x|\theta)$ is a proper distribution. The *maximum likelihood* approach selects the hypothesis $\hat{\theta}$ that maximizes $P(x|\theta)$ over the observed data. It is well known that if the set of observed data is small or noisy, the maximum likelihood $\hat{\theta}$ can generalize poorly to future examples. To address this issue, in the *maximum a posteriori* (MAP) approach a prior distribution $P(\theta)$ is defined which assigns a higher probability to "simpler" hypotheses expected to generalize well to new examples. Using a prior distribution, the MAP approach maximizes the joint probability of θ and a training set of N examples $x_1, ..., x_N$:

$$\hat{\theta} = \arg\max_{\theta \in \Theta} P(\theta) \prod_{i=1}^{N} P(x_i|\theta). \tag{2.34}$$

For numerical stability, it is common to minimize the negative log of (2.34), as this replaces a product of many small terms with a sum of larger terms. This thesis uses the terms "prior" and "regularization" interchangeably, but to be precise, the prior distribution $P(\theta)$ is a probability distribution over hypotheses, and the regularization function $\lambda\Phi(\theta)$ is proportional to the negative log of $P(\theta)$:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \lambda\Phi(\theta) + \sum_{i=1}^{N} L(\theta, x_i). \tag{2.35}$$

If the hypothesis space Θ is large, the regularization function plays the critical role of keeping implausible hypotheses from being considered until they are sufficiently supported by the data (through the empirical loss). Regularization is a mathematical version of Ockham's razor, which played a key role in the development of science by insisting that "simpler" hypotheses should be preferred over more "complex" (i.e. less simple) ones. In machine learning "simple" is defined in terms of generalization, and a critical question in successfully applying regularization is finding a function that prefers hypotheses that generalize well to new data. One widely used form of regularization is a preference for smooth functions, which can be implemented by penalizing the distance between a hypothesis and a mean vector (typically a vector of zeros). In batch (offline) learning, a regularization penalty requires that more complex hypotheses provide a greater reduction in empirical error in order to be chosen. As all of the training data is assumed to be available, batch learning minimizes a fixed objective function, and regularization influences the location of the minimum. In online learning, regularization works a little differently; it influences the order in which hypotheses are used for prediction and regularization is implemented implicitly by the update rule of the online learning algorithm.

To summarize, in *online* learning regularization determines the order in which hypotheses are tested. In *batch* learning, regularization determines how much additional training error reduction

a hypothesis needs to provide, and the order in which hypotheses are evaluated is determined by the optimization method used to minimize the objective function. This distinction is important for neural networks, as they are often trained with the online Stochastic Gradient Descent (SGD) algorithm.

Implicit Regularization in Deep Networks

Prior to 2006, attempts to learn deep neural networks with more than one or two hidden layers were unsuccessful [3], with the notable exception of convolutional neural networks [36, 37]. A typical deep neural network training methodology is given in [38]. In that work, fully-connected neural networks with one to four hidden layers were initialized randomly by drawing each weight independently from a uniform distribution over $[-1/\sqrt{k}, 1/\sqrt{k}]$, where k is the number of connections each node receives from the previous layer (the size of the previous layer since the networks were fully connected). The weights of the network were then trained with stochastic gradient descent by backpropagating gradients from a softmax output node. Section 2.1.2 discussed how in the case of linear classifiers, the SGD update rule implicitly adds a Gaussian prior (L_2 regularization function) on the weights which penalizes weight vectors that are farther from the starting weight vector. The experiments conducted by [38] found that with this training methodology, the test set error of one and two hidden layer trained networks was virtually indistinguishable, but additional layers hurt performance. For shallow neural networks learning smooth functions, the regularization provided by random weight initialization and SGD updates can work well. However, this regularization method is poorly suited to deep fully-connected networks because the L_2 regularization implicit in SGD may strongly penalize changes in the lower layers of the network.

A hypothesis proposed by [39, 40] is that gradients become less informative when they are backpropagated through more layers. It is insightful to examine why this might be the case. Stochastic gradient descent computes the gradient of the output softmax node with respect to each layer in the network, by applying the chain rule for derivatives. As shown in figure 2.2, a fully-connected neural network will have many ways (paths through the network) that a weight in an early layer can influence an output node. The chain rule computes the gradient of a particular weight based on a sum of the influences from all of these paths. When the weights of the network are randomly initialized from a zero-centered distribution, a weight early in a fully-connected network is likely to have an approximately equal number of negative and positive contributions to the output node. In fact, as the number of randomly-initialized hidden layers between the weight and the output increases, the distribution of contributions from an individual weight becomes highly peaked around zero. Figure 2.3 shows the distribution of contributions of hidden unit to the output classification computed from 1000 randomly-initialized neural networks. Note that this assumed linear hidden units. The gradient updates during training will be proportional to these contributions. With no intervening hidden layers, the distribution of updates matches the uniform distribution the weights were drawn from. As the number of intervening hidden layers increases, however, the distribution becomes peaked around zero, with small tails. Consequently, the total[8] magnitude of the gradient updates for each linear-hidden unit layer should fall off exponentially as the number of intervening layers increases (Figure 2.4).

[8] Sum of the absolute value.

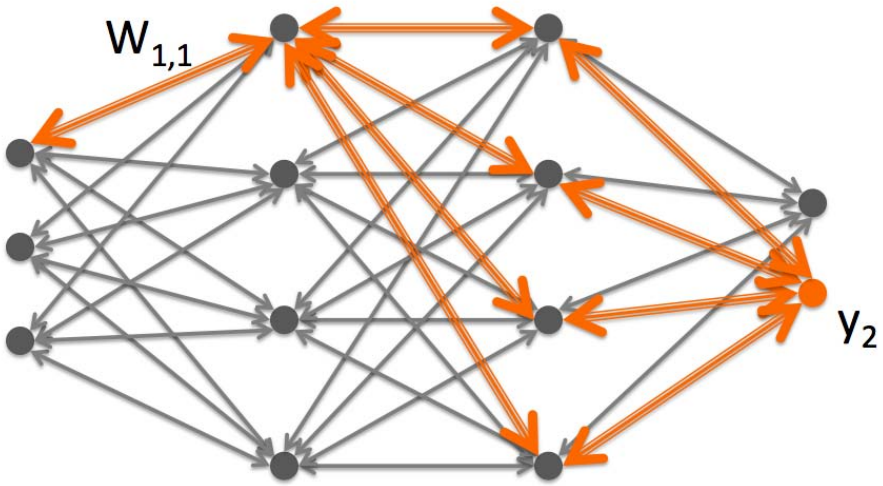


Figure 2.2: In a fully-connected neural network, weights separated from the output node have many possible paths with which they can influence the output. Here, the orange triple lines show the paths connecting the first layer weight $W_{1,1}$ to the output y_2. If the weights are randomly initialized from a zero-centered distribution, it is equally likely that individual paths will make positive or negative contributions to the output, leading to a cancelling effect that increases with the number of intervening hidden layers.

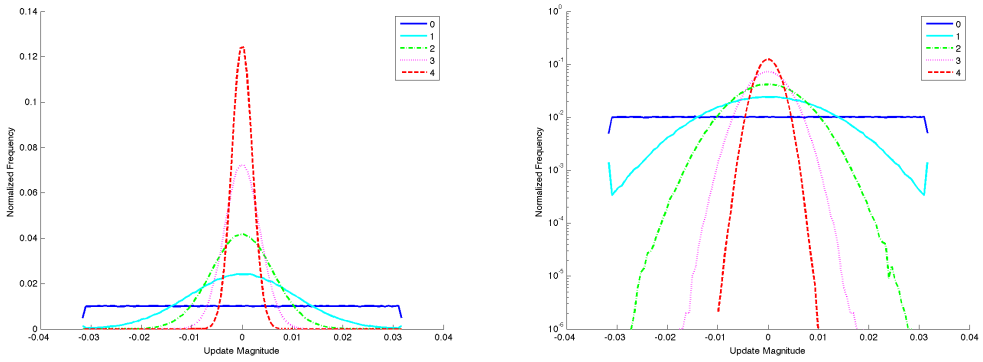


Figure 2.3: Contribution of weights at each layer.

With non-linear hidden units, such as the commonly used sigmoid and hyperbolic tangent functions, the picture becomes more complex, and an excellent analysis is given in [41]. To summarize, sigmoid hidden units have a non-zero mean and saturate at 0, which induces undesirable singular values in the Hessian and cause gradients to be squashed by the top layer, when the lower layers of the network are not computing useful features of the input. Non-symmetric activation functions such as the hyperbolic tangent do not saturate at zero and hence pass larger gradients to the lower

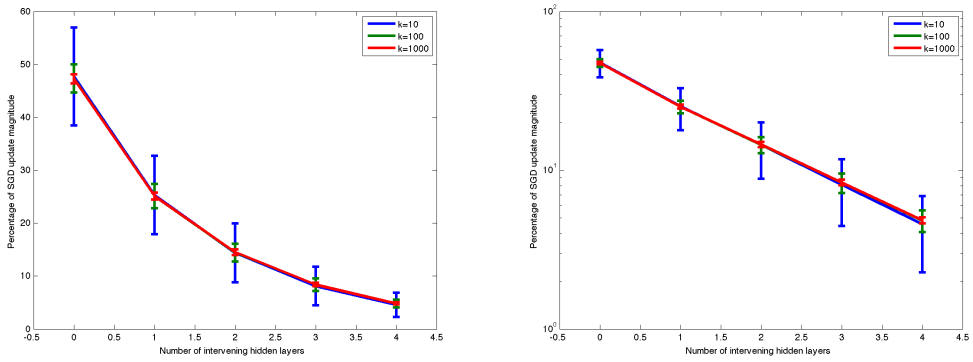


Figure 2.4: Relative size of the gradient update at each layer. Initially the layers closest to the output will capture almost all of the updates, a tendency that is independent of the size of the hidden layers.

layers, although they initially have the same averaging effect as linear hidden units, which causes the distribution of gradients to become peaked around zero in the lower layers of the network. It should be noted that even when the gradients stay large, the non-convexity of a deep network with non-linear hidden units makes it difficult to escape the basins of attraction of the local minima near the randomly chosen starting location, and it is highly unlikely that the gradients would point towards the true global minimum. Since regularization is implemented as hypothesis ordering for online algorithms, this means that the implicit SGD regularization is much stronger for lower layers of the network farther from the output. Consequently, the initial weights of the lower layers are crucial to the success of SGD regularization.

Effects of Implicit SGD Regularization

If there are d hidden layers we would expect that SGD regularization would primarily search the space of one hidden layer networks, on top of an input space of random projections formed by the bottom $d - 1$ layers. Then, only if one hidden layer is insufficient to solve the training set would the next layer be optimized. For fixed training sets, (and layers big enough to avoid optimization difficulties) SGD can be expected to learn only as deeply as it must to minimize the training set error. This results in the network learning a one or two hidden layer network on top of a random representation of the input, and it is not surprising that generalization performance is often poor once there are more than two hidden layers in the network [38]. In the online setting, SGD is prevented from converging to a local minima that does not generalize well by the constant introduction of new test examples. If the initial weights for the lower layers are far from good values, however, we would expect that strong implicit regularization on the lower layers would lead to high regret and slow learning. Indeed an online training experiment from [11] (included for reference as Figure 2.5) shows this effect. On the other hand if the lower layers start close to good hypotheses, cumulative regret during training should be much lower as only the top layers need to change. The greedy layer-wise pre-training strategy discovered by [4, 3] provides such an initialization for tasks (such as MNIST) where the generative distribution of the data is related to the target task.

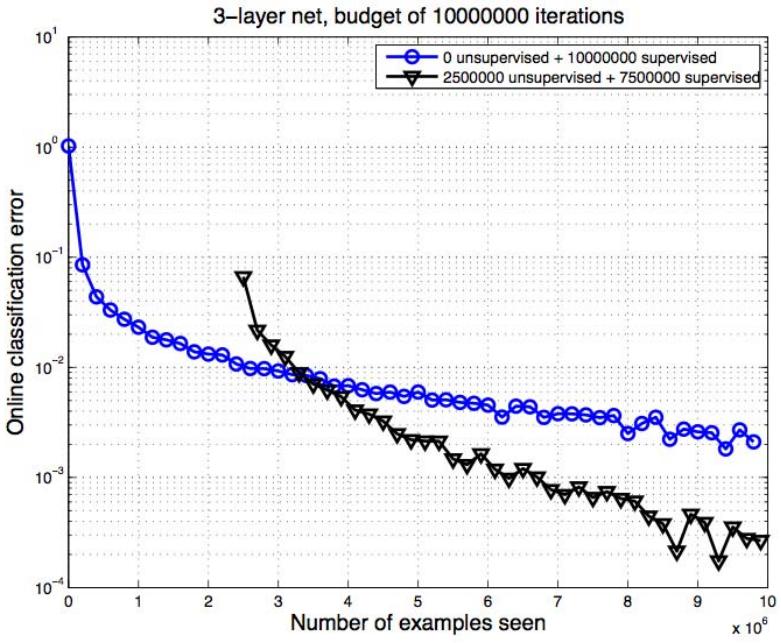


Figure 2.5: Shown is an experiment by Dumitru Erhan (taken from [11]) where deep neural networks were learned on the "infinite MNIST" data set. With data-dependent regularization through pre-training (triangles) the online classification error rate drops much more quickly than with random, uninformed regularization (circles).

The combination of SGD with random weight initialization may be ordering the hypotheses so that the space of 1-layer networks is considered first (with random projections as input) and then the space of 2-layer networks is searched (As shown in Figure 2.6). As these progressive spaces get exponentially larger, it is not surprising that training has been reported to find hypotheses (parameter vectors) for the network that have arbitrarily low training error on even large training sets, without generalizing well to new examples. This may possibly be because the first layers of the network remain as random projections of the input, and are likely to capture enough variation in the data to reduce the empirical error, without providing a feature set that generalizes well to new examples. Note that the empirical error is only an approximation to the expected error and their minimizers will not generally be the same, so when the hypothesis space is large optimizing the empirical error could find a significantly different solution than optimizing the expected error. By searching a large hypothesis space based on random projections of the input, it is not surprising that the combination of random weight initialization and SGD finds an empirical error minima that does not generalize well.

2.2.3 Alternative Regularization Strategies

We have seen how the regularization implicit in standard neural network learning techniques (SGD from random weights) may create an ordering on the hypothesis space that is particularly ill-suited

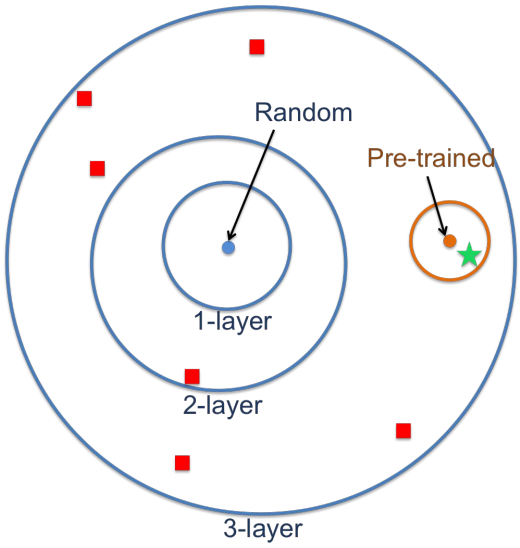


Figure 2.6: Diagram of the hypothesis space consisting of the possible parameter values of a 3-layer neural network. Stochastic gradient descent training from random initialization performs a local search in the hypothesis space that is ordered to optimize the top layer first, and then the next lowest layer.

to deep architectures; poor initialization of the lower layers leads to high regret in the online case and poor generalization in the batch case. Therefore, a key challenge in deep learning is improving the regularization of the lower layers by finding a better initialization for the SGD algorithm. For vision applications it has been found that the sparse local weight connections between layers and the weight tying provided by convolutional neural networks [37] can be a useful regularization tool that produces state of the art generalization performance, but what can be done for general applications?

Input Reconstruction

In 2006, it was discovered that adding an input-modeling pre-training step to the early layers of the neural network could dramatically improve generalization performance [4, 3] on some tasks. In this approach, each layer of the network is trained to reconstruct its input, by optimizing an unsupervised generative model such as an autoassociator or a Restricted Boltzmann Machine (RBM). Once the weights have been optimized, the output of the layer is used as input to a generative model on the next highest layer. After this pre-training phase the weights are trained by SGD as usual. The black line in Figure 2.5 shows the result of applying this pre-training method to the online infinite-MNIST problem, and it shows the fast convergence expected when SGD is only required to optimize the top layers of a deep network.

This input-reconstruction pre-training step finds parameter vectors (hypotheses) that capture regularities in the data, and corresponds to maximizing $P(X)$, the probability of the unlabeled input data. By replacing the random initialization with this pre-trained initialization–thereby shifting the starting point of SGD–greedy layer-wise pre-training provides data-dependent regularization that is

particularly effective for many tasks. Instead of trying to learn a 1-layer network on top of random projections of the data, gradient descent starts from a representation of the data that has been trained to capture statistical regularities in the data.

Auxiliary tasks and local information

The SGD training algorithm is *local* because it has limited ability to optimize weights that are separated from the output by multiple non-linearities, or that make conflicting contributions to the output. In the absence of an alternative to SGD, alternative sources of information–like input reconstruction–must be found that are local to early parts of the network. Ideally these local information sources or *auxiliary* tasks [42], will select parameters for early modules that are also useful for the *target* prediction task. Unsupervised pre-training provides an input-reconstruction auxiliary task that is related to many target prediction tasks, but it is not the only auxiliary that has been shown to be useful. A variety of auxiliary tasks for image and text applications were introduced in [42] that involve holding out parts of unlabeled input vectors, and predicting functions of the held out inputs from the rest of the input vector. For natural language processing application, [43] provides impressive results using a set of auxiliary tasks at various points in a deep architecture. Embedding algorithms [44] can also be an effective auxiliary tasks for many applications. Even weighted versions of the training set used for the target task can serve as effective auxiliary tasks. In the curriculum approach [45], each weighted training set serves as an auxiliary task.

The majority of this thesis is devoted to developing learning modules and auxiliary tasks that provide local information. First, though, the next chapter will examine a problem where a deep modular architecture has proven to be quite effective: mobile robot navigation.

Chapter 3

Mobile Robot Perception

Figure 3.1: The "Crusher" robot pictured above provides state of the art autonomous driving capabilities in unstructured off-road environments, by interpreting data from onboard sensors and previously-collected overhead imagery.

Autonomous driving in unstructured environments is a key capability for applying robotics in areas such as agriculture, mining, forestry and military operations. It also provides a prime example of a complex problem that benefits from a deep modular solution. Much of this thesis was inspired by challenges encountered when developing the "Crusher" robot, pictured in Figure 3.1, as part of the UPI project. Crusher was tasked with driving autonomously to a goal location several kilometers away through an unstructured environment while using data from onboard sensors to determine a safe, efficient path. This task can be divided into two components, a perception component that translates the raw sensor data into feasible paths through the environment, and a control component that actuates the vehicle to follow a path. The perception component is complicated by the high-dimensional nature of the input data and the need to produce structured output in the form of continuous paths. Past work in mobile robot navigation has discovered that dividing the perception

problem into a network of modules can work particularly well.

Early work in autonomous off-road driving included Terragator [46], Navlab [47], and the Autonomous Land Vehicle (ALV) [48]. These systems defined key challenges and pioneered approaches such as combining a terrain classification step with a global planning algorithm such as A* [49]. Later research with NAVLAB II [50] improved performance by introducing the D* planning algorithm which could efficiently replan a path to the goal given classification updates provided by the robot's onboard sensors. Through extensive additional research in the DEMO III, Robotics Collaborative Technical Alliance (RCTA), and PerceptOR [51] program an effective high-level design for mobile robot navigation systems emerged: sensor data from each 2-D cell in an environment are used to produce a numeric cost for that cell, a global-scale planner is used to plan a path to the goal, and a local planner more informed about the dynamic constraints of the vehicle is used to approximately match the global plan with a realizable path the robot can follow.

The final PerceptOR perception system [51] contained a large number of parameters for the various modules that often required time-intensive manual tuning to operate well in new environments. Consequently a major goal of the UPI program was to reduce manual parameter tuning by using as much learning as possible in the system. Several systems for autonomous driving have been entirely learning-based. The Autonomous land vehicle in a neural network (Alvinn) [52] was able to follow roads by training a neural network to directly predict the steering angle used by a human driver given an image from a camera viewing the road. Later the DAVE system [53] demonstrated autonomous off-road driving and obstacle avoidance using a deep convolutional neural networks architecture. Like Alvinn, the system was trained to directly predict the steering angles from raw images (in this case a stereo pair of cameras). However, both systems were unable to produce goal-directed driving behavior due to the lack of a planning module.

3.1 UPI mobile robot perception system

This thesis was inspired by challenges encountered while developing the UPI perception system for off-road autonomous mobile robot navigation. A simplified view of the final UPI perception system[1] is shown in Figure 3.2. This system extends the approach pioneered in the PerceptOR program and outlined in [51]. Overhead data–typically satellite imagery–is used for long range planing through portions of the environment that the robot has not observed. This pathway can be trained to imitate human-drawn example paths [5, 22], or to predict the output of short-range perception system and generalize it to unseen areas [55]. As the robot drives, a set of laser range finders (Ladar[2]) and cameras provide input to the long-range and short-range perception systems. The appearance of terrain to the sensors changes with range, and to simplify the learning problem, long-range perception is trained to predict the output of the short-range perception system on areas that have been observed at both short and long ranges [55]. As it serves as "ground truth" for the long-range and overhead perception systems, accurate short-range perception is critical.

The short range perception system takes as input 3-D points from the ladar that have been projected into camera images, and tagged with local properties of the image such as color, and image texture. The local perception system then discretizes the space surrounding the robot into a

[1] For clarity some perception systems, such as ground plane estimation [54], have been omitted.

[2] LAser Detection And Ranging is a high-fidelity version of radar that uses light instead of radio waves.

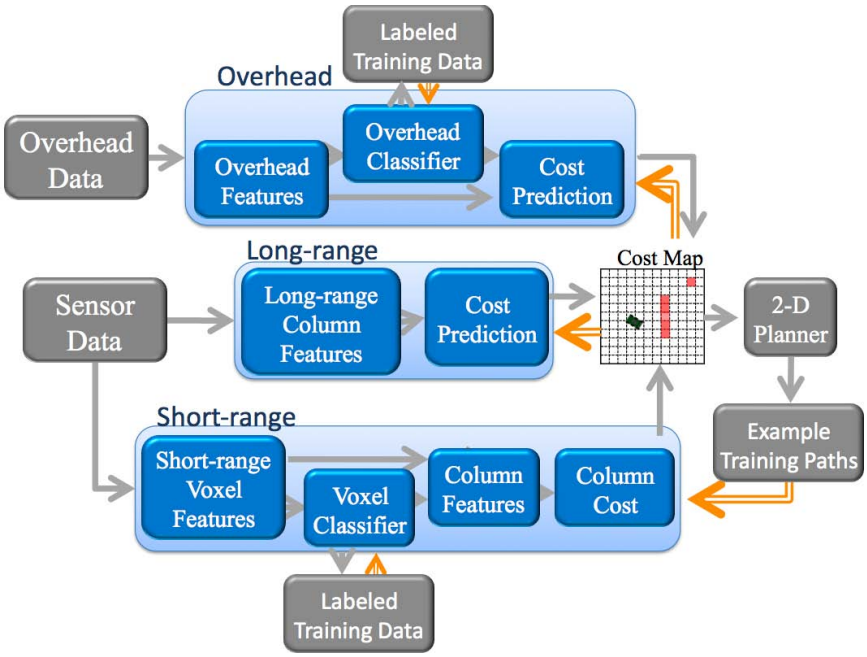


Figure 3.2: Start-of-the art mobile robot navigation solution used by the UPI program. The mobile robot navigation problem has been broken down into a series of modules, and the above diagram visualizes the information flow between the modules in the system. Only the voxel classifier uses any machine learning, and the parameters of the other modules were painstakingly optimized by hand.

3-D grid of voxels[3] and computes features of the tagged laser points over each voxel. Examples of some of these engineered features are shown in Figure 3.3, and they include the averages of the point tags, eigenvalues of the local point cloud scatter matrix [56], the surface normal (3rd eigenvector of the scatter matrix), and the probability and strength of laser pulse reflection from the voxel. A voxel classification module then classifies each voxel containing a minimum number of 3-D points into "ground", "vegetation", or "obstacle" classes. Next a cost function reduces the 3-D grid of voxel classifications to a 2-D "cost map" containing costs for each column of 3-D space. A planning algorithm sends the minimum cost path through the cost map to the vehicle control portion of the autonomy system, and the cost values produced by the local perception system also serve as training examples for the long-range and overhead perception systems. Various interpretations have been proposed for the meaning of the cost values, such as mobility risk, but due to the tightly coupled nature of mobile robot systems they have no fundamental interpretation apart from the paths that they cause the planner to produce through the environment. My work on the PerceptOR and UPI programs involved key contributions to the short-range perception system that increased its accuracy, efficiency and robustness; they are outlined in the remainder of the chapter.

[3] A volume element or "voxel" is the 3-D equivalent of a pixel.

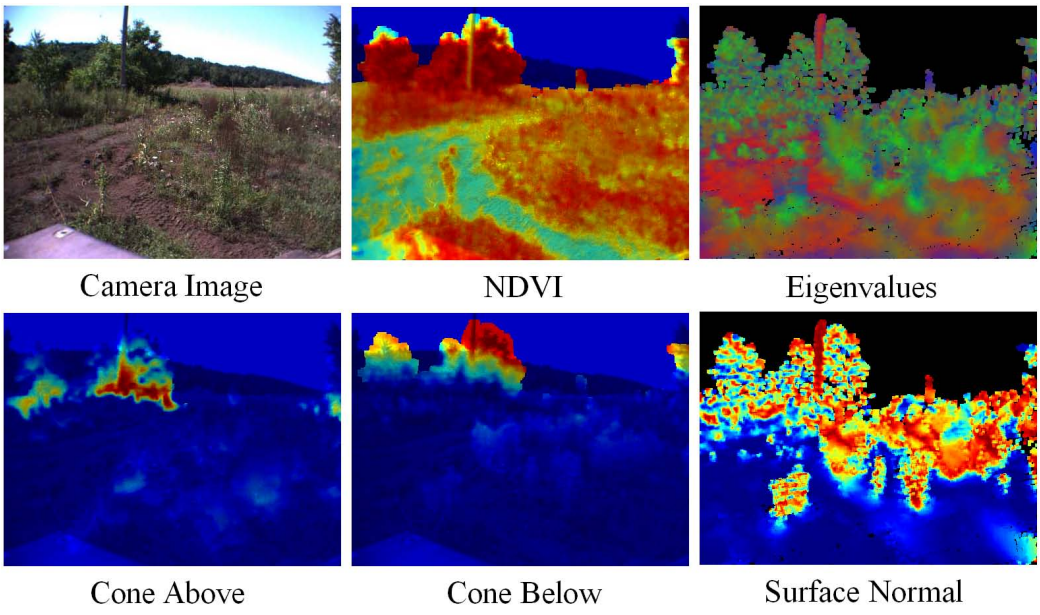


Figure 3.3: The camera and laser data captured by the sensors onboard the mobile robot are used to compute a set of engineered features. Here a few of the features are shown projected into the image from one of the onboard cameras.

3.2 Range Image Voxel Classification

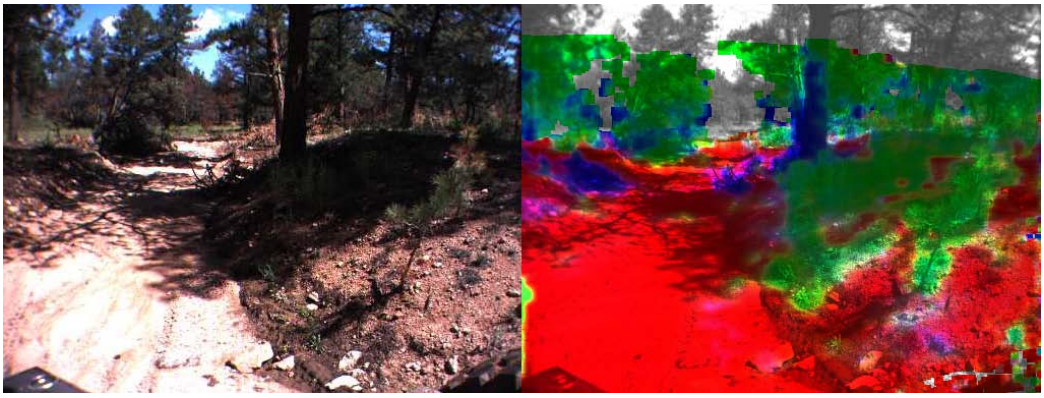


Figure 3.4: Classified range image voxels in a forest scene.

Previous work on mobile robot perception [51, 57] accumulated statistics from 3-D points in a cartesian grid of the local area which remained fixed relative to the earth. This approach has many advantages, but one serious drawback is that the cartesian grid does not match the sampling pattern of sensors on the robot. The camera and ladar sensors on the vehicle have a constant angular sampling rate, and the distance between measurements increases as the distance from the vehicle

increases. The result is that voxels in the grid close to the robot are sampled heavily, and voxels far from the robot receive few, if any, samples. Because of this sampling effect, choosing a fixed grid resolution becomes problematic. Geometric features such as the scatter-matrix eigenvalues require a minimum number of points to achieve stable values, forcing a trade-off between stable feature values far from the robot and fine spatial resolution close to the robot.

An innovation of my work was to store points in a spherical grid or "range image" centered on the current location of the robot. A spherical grid matches the sampling pattern of cameras and Ladar sensors: the size of the voxels increases with the distance from the vehicle, meaning each spherical voxel has a roughly constant number of points, allowing features like scatter-matrix eigenvalues which depend on having a certain number of points to be computed reliably and efficiently at longer ranges. The spherical grid does have the disadvantage of requiring 3-D points to be re-projected into the grid when the robot moves, however for the Crusher robot this was actually advantageous. The six SICK ladar sensors on Crusher together acquire over 100,000 3-D points a second and maintaining only the last 1-2 seconds of scanning proved sufficient for accurate classification. This meant that each point was projected into the grid only 4-8 times before it expired from the buffer. Additionally, because Crusher is a skid-steer robot, accurate pose estimation proved challenging and automatically expiring quickly reduced the effect of pose estimation problems.

Features computed using the spherical grid were then classified with a multi-class logistic regression (maximum entropy) classifier into "ground", "vegetation", or "obstacle" classes, using a labeled data set of over 4 million voxels collected from a variety of test environments. This "range image classifier" (Figure 3.4) provided state-of-the-art terrain classification performance during hundreds of kilometers of autonomous testing, due in part to advances in vegetation detection [7, 58].

3.3 Vegetation Detection

Autonomous off-road navigation presents unique perception challenges. In environments such as hallways and on roads, it suffices to assume that all objects are rigid, and avoid anything that is not part of the ground plane. In off-road driving, however, the assumption that every object is a rigid obstacle that the robot must avoid quickly presents problems. In situations such as a field of tall grass, there may be dense objects on all sides of the robot. In order to plan safe, efficient paths through such an environment, the robot must be able to reliably discriminate between vegetation that it can drive through if necessary, and rigid obstacles such as tree trunks and rocks that can cause damage (Figure 1.3). For safe high-speed operation performing this discrimination at range becomes increasingly important.

Methods have been developed to detect vegetation from 3-D point clouds [56, 59]. At longer ranges the limited viewpoint of onboard sensors, reflection of the laser pulses away from the scanner and laser beam divergence make it difficult to obtain point clouds of sufficient quality and density. Fortunately there are well-established techniques for measuring chlorophyll content using a multispectral camera [60, 61, 62, 63, 64, 65, 66, 67] that have been developed for satellite-based remote sensing. A simple pixel-by-pixel comparison between red and Near-InfraRed (NIR) reflectance, normally referred to as a vegetation index or a band-ratio, provides a powerful and robust way to detect vegetation. Further, most CCDs have significant spectral response all the way out to

wavelengths of around 1000 nm, meaning that a standard monochrome CCD can be converted into a NIR camera simply by covering it with a longpass filter. Although the viewpoint of a satellite is drastically different from that of a mobile robot, this technique has proven to be effective despite additional complications such as views of the sky, or shadowed areas that are lit by both light reflected off of other surfaces and light reflected from the sky.

Little prior work in the mobile robotics community used multi-spectral information for ground-based terrain classification for navigation. An early attempt, [68], used data from a filter wheel camera to label pixels as chlorophyll-rich vegetation or soil based on thresholding the ratio of the NIR and red values. Later work in [69] mentioned the usefulness of NIR in detecting photosynthetic vegetation, but described the use of a Gaussian mixture model-based classifier with only RGB features. Aside from the overall speed of the classifier, no quantitative performance analysis was given, and the role played by the features in the larger system was largely presented as anecdotal.

This work [58, 7] provides an extensive qualitative evaluation of several vegetation indices across different environments and geographic locations, and quantitatively evaluates the utility of combining vegetation indices with 3-D data. In particular it evaluates combinations of popularly used shape, density and color features and shows the effect of augmenting these features with NIR and NDVI information on a terrain classification task. This analysis measures the reliability of different feature sets with respect to sensing range and demonstrates through a set of experiments that vegetation indices have properties complementary to the color and shape descriptors traditionally used for point cloud analysis. Adding vegetation indices leads to significant improvement in classification performance for tasks relevant to outdoor navigation.

3.3.1 Approaches to Vegetation Detection

This work combines two complementary approaches to vegetation detection, vegetation indices (also referred to as band-ratio techniques) that have long been used in the remote sensing community, and more recent 3-D structure based techniques from the mobile robotics community.

Vegetation Indices

The spectral properties of chlorophyll-rich vegetation are primarily determined by the absorption spectra of water and chlorophyll, and the refraction of light at cell walls [70]. The water present in cells absorbs light with wavelengths longer than 1400 nm. Chlorophyll strongly absorbs visible light, especially red and blue wavelengths [60]. The remaining light is efficiently scattered by the critical internal reflection caused by the change in refractive index from water to air at the cell wall. As a result, those wavelengths between 700 nm and 1400 nm that escape both water and chlorophyll are strongly reflected in all directions.

The sharp difference between the reflectance of vegetation at 645 nm (red) and at 780 nm (NIR) has long been exploited in the field of satellite remote sensing. Kauth and Thomas [64] noticed that plotting NIR reflectance against red reflectance for satellite images produced a scatter diagram with a line of points formed by pixels containing bare soil, and a cluster of points from pixels completely covered with vegetation. Points with a mixture of vegetation and soil appear between the soil line and the vegetation point. Figure 3.5 shows this scatter plot created from a typical image. Because the camera also includes a view of the sky, the scatter plot contains a blue sky region (bottom

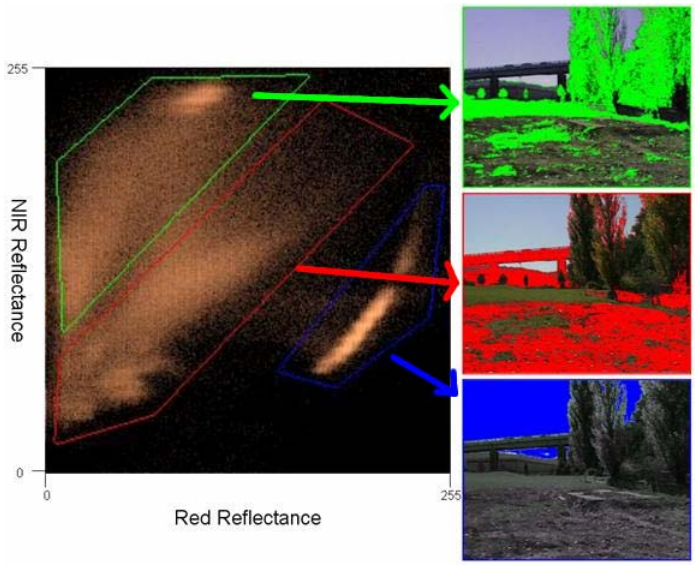


Figure 3.5: Scatter plot of NIR reflectance vs. red reflectance for all pixels in a typical image. Different regions in the scatterplot correspond to different types of materials in the image. Pixels in the green region correspond to vegetation (top image), pixels in the red region are mainly soil and man-made structures (middle image), and pixels in the blue region correspond to sky (bottom image).

image, marked in blue) as well as the soil region (middle image, marked in red) and the vegetation region (top image, marked in green). Clouds blend into the soil line, but are still very distinct from vegetation. Pixels containing vegetation and blue sky are remarkably well separated from everything else in a natural scene.

One of the most popular ways to use the information contained in the red and NIR bands for remote sensing applications is to compute a quantity known as the Normalized Difference Vegetation Index (NDVI) which varies from -1 (blue sky) to 1 (chlorophyll-rich vegetation) [65].

$$\text{NDVI} = \frac{\rho_{\text{NIR}} - \rho_{\text{RED}}}{\rho_{\text{NIR}} + \rho_{\text{RED}}} \tag{3.1}$$

Several attempts have been made in the remote sensing literature to correct deficiencies in this index [67, 62, 71, 72], particularly in shadows and underexposed areas. Since the NDVI measures the slope from the origin of the red-NIR space, sensor noise and errors in the radiometric calibration of the red and NIR sensors have a much greater effect in underexposed areas. Shadows are particularly challenging, since the reflected light that illuminates shadowed regions can have a spectral distribution that is significantly different from that of sunlight, usually shifted towards blue wavelengths because of atmospheric scattering.

The typical use of NDVI in remote sensing is to measure the Leaf Area Index (LAI), the percentage of the ground surface that is covered by vegetation. However, when the goal is linear classification into vegetation and non-vegetation categories, it is useful to provide the raw NIR and red values as well as the NDVI to the classifier. This is because a linear decision boundary based

solely on the NDVI corresponds to a line intersecting the origin of the red-NIR space, whereas a linear classifier operating on the raw pixel values and a constant bias feature can produce a decision boundary with an arbitrary intercept. NDVI is discussed to tie this vegetation detection technique back to its origins in the remote sensing community, but the actual classifiers evaluated in section 3.3.2 use the raw pixel values as well as the NDVI, since for a linear classifier they may be more informative than using many of the variants of the NDVI.

Validation of Approach

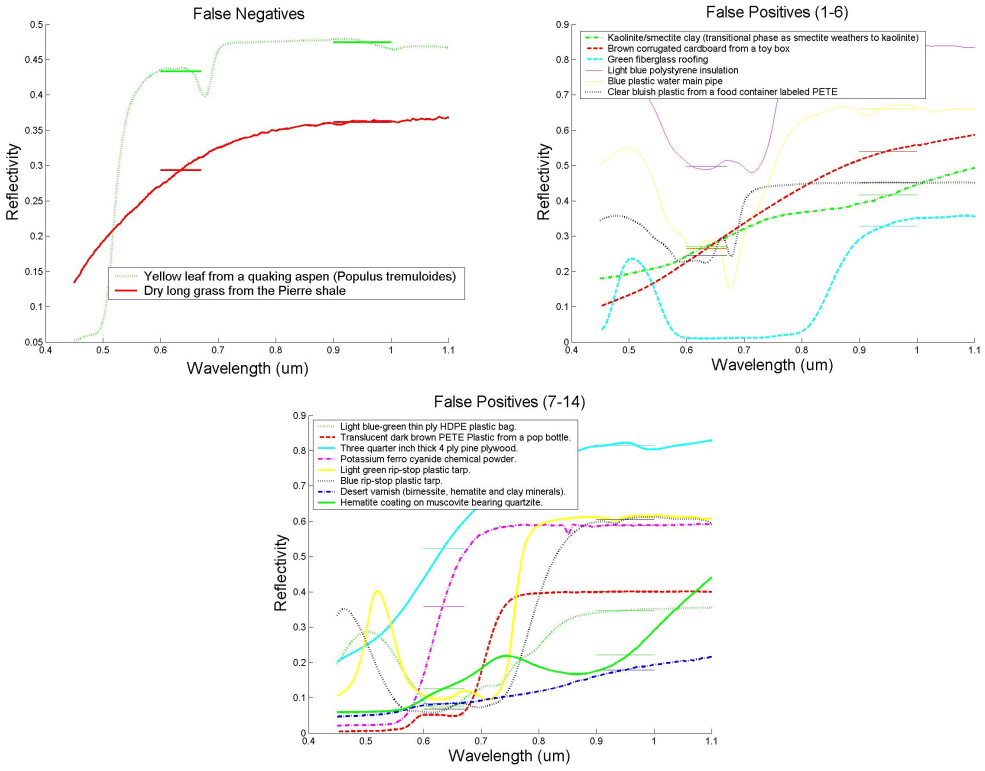


Figure 3.6: Material spectra misclassified by an optimal linear discriminator using the red and NIR bands (600-670 nm and 900-1000 nm). Horizontal line segments indicate the simulated camera responses used for classification. 258 of 274 common materials are classified correctly. The failures are mainly dead vegetation where the chlorophyll has had time to degrade, man-made materials such as plastic, and natural mixtures involving hematite.

The USGS digital spectral library [73], provides a useful tool for selecting an appropriate NIR filter, and verifying the effectiveness of using NIR in conjunction with red across a variety of different types of vegetation and non-vegetation. This library contains the spectral signatures of over 800 different materials. Many of these spectra are from rare minerals that are not relevant to our application. The relevant subset of the library used for testing contained 105 common types of vegetation and 169 common soil mixtures, artificial materials, and coatings. The optimal linear separator on

this data set classifies 258 of 274 materials correctly. A yellow aspen leaf and dry grass are misclassified as non-vegetation, and various man-made materials and natural mixtures involving hematite are misclassified as vegetation (figure 3.6).

Figure 3.7: Small images show the RGB and NIR appearance of two scenes. Large images are the RGB image with areas of high NDVI highlighted in bright green. While generally reliable in natural scenes, NDVI can fail on synthetic materials such as the paint on the truck in the right image. In this case 3-D methods would easily classify the side of the truck as a rigid obstacle instead of vegetation. Note that NDVI correctly classifies the brown dead grass in the left image as vegetation.

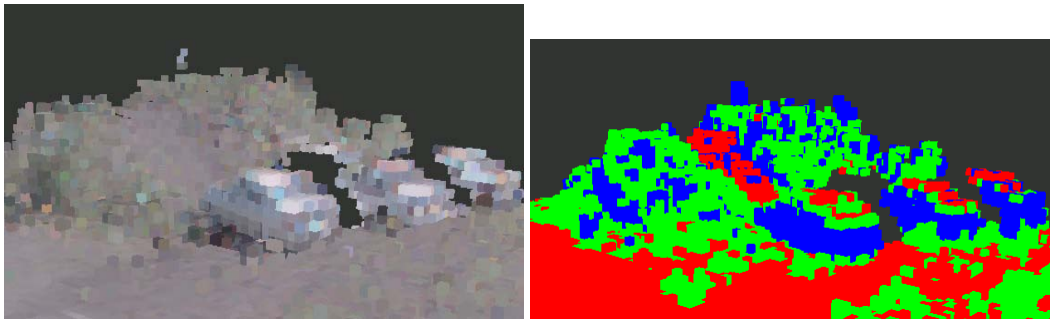


Figure 3.8: Left: average RGB values of each voxel in a scene containing cars. Right: after 3-D classification the voxels containing the flat sides of the cars are classified as obstacles (blue). Voxels containing curved car surfaces have more of a vegetation-like signature (green), and voxels corresponding to ground are marked in red.

3-D Methods

A complementary approach to vegetation detection, presented in [56], uses the spatial distribution of the local ladar point cloud to classify the region into surfaces, linear structures, and a class referred to as scatter, which includes tree canopy and porous vegetation. This method first computes the eigenvalues of the covariance matrix of the local point cloud (defined as all points within a certain distance of the point of interest), and then classifies the point based on the relative magnitudes of those eigenvalues. Linear structures have one dominant eigenvalue. Surfaces have two large eigenvalues, and an area is declared to be scatter when the third largest eigenvalue is a significant fraction

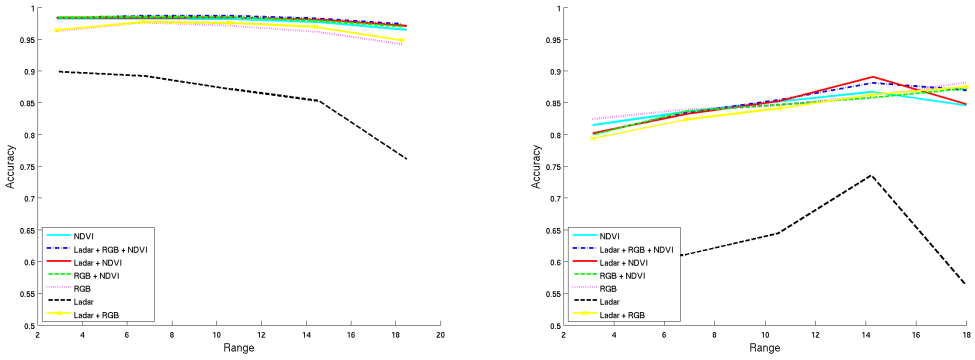


Figure 3.9: Voxel classification accuracy vs range for telling rigid obstacles (rocks, tree trunks, telephone poles, cars, etc...) apart from non-rigid obstacles (grass and bushes). Left: training set performance. Right: Test set performance. Inclusion of RGB or NDVI features provides a significant performance boost over ladar features alone.

of the largest eigenvalue. In addition, the estimated surface normal of the local area, another useful feature of the local point cloud, is recovered by this computation as it is simply the eigenvector corresponding to the smallest eigenvalue of the covariance matrix.

This 3-D method performs particularly well on certain man-made structures where the NDVI approach is known to fail. For instance, certain types of vehicle paint give off a vegetation-like NDVI signature (figure 3.7), but the flat sides of vehicles are easily detected as surface 3-D structures (figure 3.8). However, the 3-D method does require a relatively dense, high-quality point cloud, which limits its application to areas closer to the robot.

3.3.2 Voxel Classification Experiment

The voxel classification component of the mobile robot perception system described in Section 3.1 was trained and tested using data from two significantly different physical environments. Training examples of rocks, bushes, grass, trees, and man-made structures such as cars, telephone poles, and barrels were gathered from a site in Western Pennsylvania. More examples of rocks and bushes were collected several weeks later in natural terrain in the foothills of the Colorado Rockies. Voxels were labeled by either hand-labeling the point cloud, or in some cases labeling everything the vehicle drives over. In both cases the features (density, surface normal, scatter matrix eigenvalue, RGB, NIR, and NDVI) of each labeled voxel were recorded every time a new laser point in that voxel was observed.

3.3.3 Classification Results

We first compare the performance of the feature sets for similar (but physically separate) environments and lighting conditions. Figure 3.9 compares classifiers trained for the binary task of discriminating between non-rigid voxels (grass, bushes, etc...) and rigid voxels (Tree trunks, rocks, cars, telephone poles, bare ground, etc...). Both the RGB and the NDVI features can accomplish

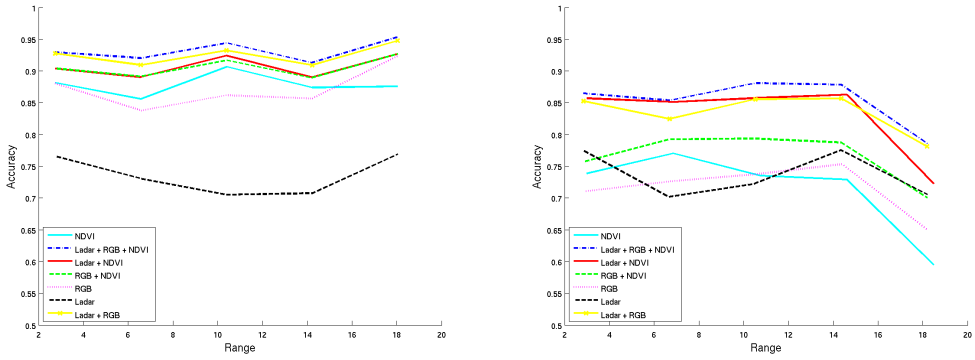


Figure 3.10: Voxel classification accuracy vs range for discriminating between rigid obstacles, non-rigid vegetation, and ground. Left: training set performance. Right: Test set performance. This task, which is used in ground plane estimation, benefits significantly from combining ladar and RGB or NDVI features.

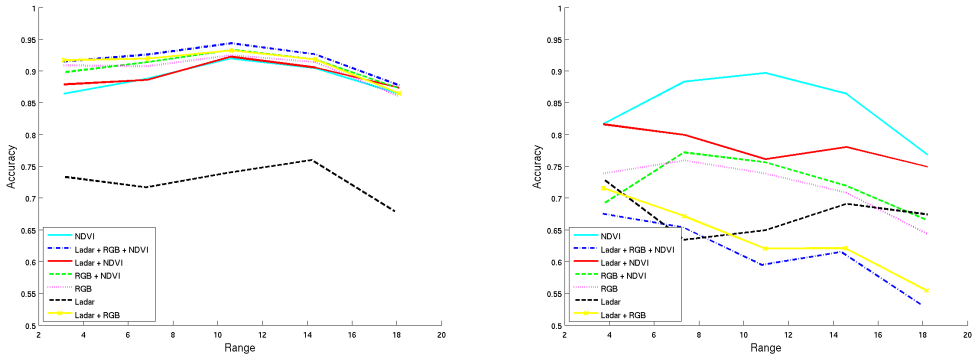


Figure 3.11: Voxel classification accuracy vs range for discriminating between rocks and bushes. Left: training set performance. Right: Test set performance. The training set for this task was collected in Pennsylvania, and the test set was collected in Colorado. The NDVI features show superior generalization to the novel environment.

this task successfully, showing roughly a 20% improvement in classification accuracy over ladar features alone. The ladar features become crucial, however, in the more complicated three-way classification task used by the robot to estimate the true ground plane of the scene. This task is similar to the previous task, with the exception that the rigid voxel class is divided into an obstacle class and a road class (horizontally oriented bare ground surfaces). In Figure 3.10 we see that combining the camera features with the ladar features boosts the total classification accuracy by approximately 10%. This performance boost is almost entirely from improvements in the ability to discriminate between the obstacle and road classes, as shown in Tables 3.2 & 3.3. Again in this task we see that the camera features are very helpful in discriminating between non-rigid voxels and the other two classes (Table 3.1).

	True Class		
Predicted Class	obstacle	vegetation	ground
obstacle	88.7	11.4	3.1
vegetation	9.5	64.0	27.9
ground	1.8	24.6	68.9

Table 3.1: Confusion matrix for a classifier trained only with ladar features

	True Class		
Predicted Class	obstacle	vegetation	ground
obstacle	88.1	0.2	4.7
vegetation	6.4	95.1	21.2
ground	5.5	4.7	74.1

Table 3.2: Confusion matrix for a classifier trained with all features

Finally we investigate the generalization ability of the different feature sets across different geographic environments on the task of discriminating between rocks and bushes. For this test the training set is from Pennsylvania, and the test set is from Colorado. As shown in figure 3.11, the NDVI features are hardly affected by the change in environment, and their performance degrades only slightly. The RGB features, on the other hand, allow the classifiers to over-fit to the specific lighting and flora of the training set, leading to vastly degraded performance in the novel environment.

	True Class		
Predicted Class	obstacle	vegetation	ground
obstacle	78.7	0.2	21.1
vegetation	6.7	95.1	21.4
ground	14.6	4.7	57.5

Table 3.3: Confusion matrix for a classifier trained with only camera features

Chapter 4

Learning from a Conditional Distribution

Pre-training an intermediate module by learning from a related auxiliary task defined by a labeled data set can be effective. However, this method assumes that the joint distribution $P(X, Y)$ of the auxiliary task is related to the target task that the module is trying to solve. In many situations in robotics the conditional distribution $P(Y|X)$ may be more useful than the joint distribution $P(X, Y)$ because the distribution of examples $P(X)$ is actually a mixture of the distribution of each environment (terrain, season, lighting condition) $P_i(X)$ that the labeled data has been collected from. Training the module to minimize the average error on the labeled training set is equivalent to assuming that the module will encounter a random ordering of examples drawn from all of the environments in $P(X, Y)$. However, in practice the environment the robot is operating in often changes slowly, and it may be advantageous to adapt the module to optimize performance on the current environment.

The "Crusher" robot described in Chapter 3.1, has operated in many different terrain types and seasonal conditions. In each different environment, sensor data has been collected and labeled to make sure that the perception system is operating correctly. As shown by the randomly selected training set images in Figure 4.1, over the life of the program this has resulted in data from a diverse set of terrain, seasons and weather/lighting conditions. After the robot had operated in a few different environments, it was observed that training the voxel classifier on labeled data collected from certain environments was hurting performance on other environments, and it became necessary to selectively drop some training data from the training set in order to improve voxel classification accuracy on the environment in which the robot was currently operating.

Table 4.1 displays the difference in voxel classification accuracy between using a labeled training set drawn from the other environments the robot has operated in and one drawn from the current environment the robot is operating in. It shows that in many cases the performance of the voxel classifier on a particular environment is significantly handicapped by training on labeled data from the other environments.

The difference between environments can also be seen directly in terms of the voxel feature vectors. Figure 4.2 shows the result of projecting the feature vector associated with each labeled voxel into the most discriminative 2-D subspace, as computed by Linear Discriminants Analysis (LDA) to separate "road", "vegetation" and "obstacle" classes. Each plot shows the labeled voxel

Figure 4.1: Randomly-sampled images from the the labeled voxel dataset collected for the UPI program reflect the wide variety in environments in which the Crusher robot has operated. Different regions of the country, and even different times of day, can produce substantially different distributions of voxels. As shown in Table 4.1, minimizing training error over all the environments is not as effective as training only on environments similar to the one the robot is currently operating in.

Environment				Training Set Drawn From:	
Name	Location	Season	Terrain	Others	Current
Taylor	PA	Winter & Snow	Woods & Grassland	71.8%	92.0%
Gascola	PA	Summer	Woods & Grassland	70.3%	94.3%
Ft. Bliss	TX	Winter	Scrubland	74.1%	98.5%
Sommerset	PA	Spring	Woods & Grassland	93.0%	95.5%
Ft. Drum	NY	Summer	Woods & Grassland	97.6%	98.7%
Gascola	PA	Winter	Woods & Grassland	94.7%	95.6%
Willow St.	PA	Fall	Urban	88.3%	98.9%

Table 4.1: It is possible to dramatically improve the accuracy of the voxel classifier component of Crusher's perception system by training its parameters only on data that is relevant to the environment the robot is currently operating in. This table compares classification accuracy in several different environments when the voxel classifier training set is drawn from the current environment with the accuracy when the training set is drawn from other environments.

data from a particular environment as points colored by class. The decision boundaries of the optimal maxent[1] classifier for each environment are shown as black lines. These optimal decision boundaries can change substantially between environments.

4.1 Automatic Adaptation From Unlabeled Data

The environmental variability faced in problems such as outdoor autonomous driving can be mitigated somewhat by taking advantage of the stream of unlabeled data the robot receives from its current environment. With certain assumptions, this unlabeled data can be used to allow the robot to automatically adapt its training set to match a test environment without requiring human intervention to label additional examples or decide which examples should be dropped from the training set. The unlabeled examples collected continuously by the robot are automatically used to estimate an importance weight for each example in the training set that will cause the training set to better approximate the test environment. The classifier is then retrained to minimize error over this reweighted version of the training set. This approach is an example of importance sampling and is often referred to as adapting to "covariate shift" [74], and it bears similarities to methods such as [75] which use unlabeled data to train a simple classifier that is locally accurate in a complex, non-linear space. By making two simplifying assumptions, the joint distribution in the labeled test set, $P^*(Y, X, \text{test})$ can be approximated with a weighted version of the training set.

4.1.1 Assumption #1: invariant conditional probability

The first assumption is that given the input features X_i, the conditional probability distribution of the class label Y_i of voxel i in the labeled test set, $P^*(Y_i|X_i, \text{test})$, is the same as the conditional distribution in the labeled training set $P^*(Y_i|X_i, \text{train})$:

$$P^*(Y|X, \text{test}) = P^*(Y|X, \text{train}). \tag{4.1}$$

This assumption implies that a change in the prior distributions $P^*(X_i|\text{train})$, and $P^*(X_i|\text{test})$ explains the change in the joint distribution of X and Y between the test and training sets. In the autonomous driving problem, this is equivalent to assuming that the definition of the rigid and non-rigid classes does not change between environments, only the relative distribution of those classes. This assumption is not always true, but the results show that there is still significant value in modeling the shift of the prior.

Using the assumption of invariant conditional probability, importance sampling can be used to weight examples in the training distribution so that the weighted loss over those examples approximates the expected loss on the test distribution. Starting with the definition of the expected loss on the test set (4.2), multiply by 1 and rearrange as in standard importance sampling (4.3) [74].

[1] Maximum Entropy, also known as multi-class logistic regression.

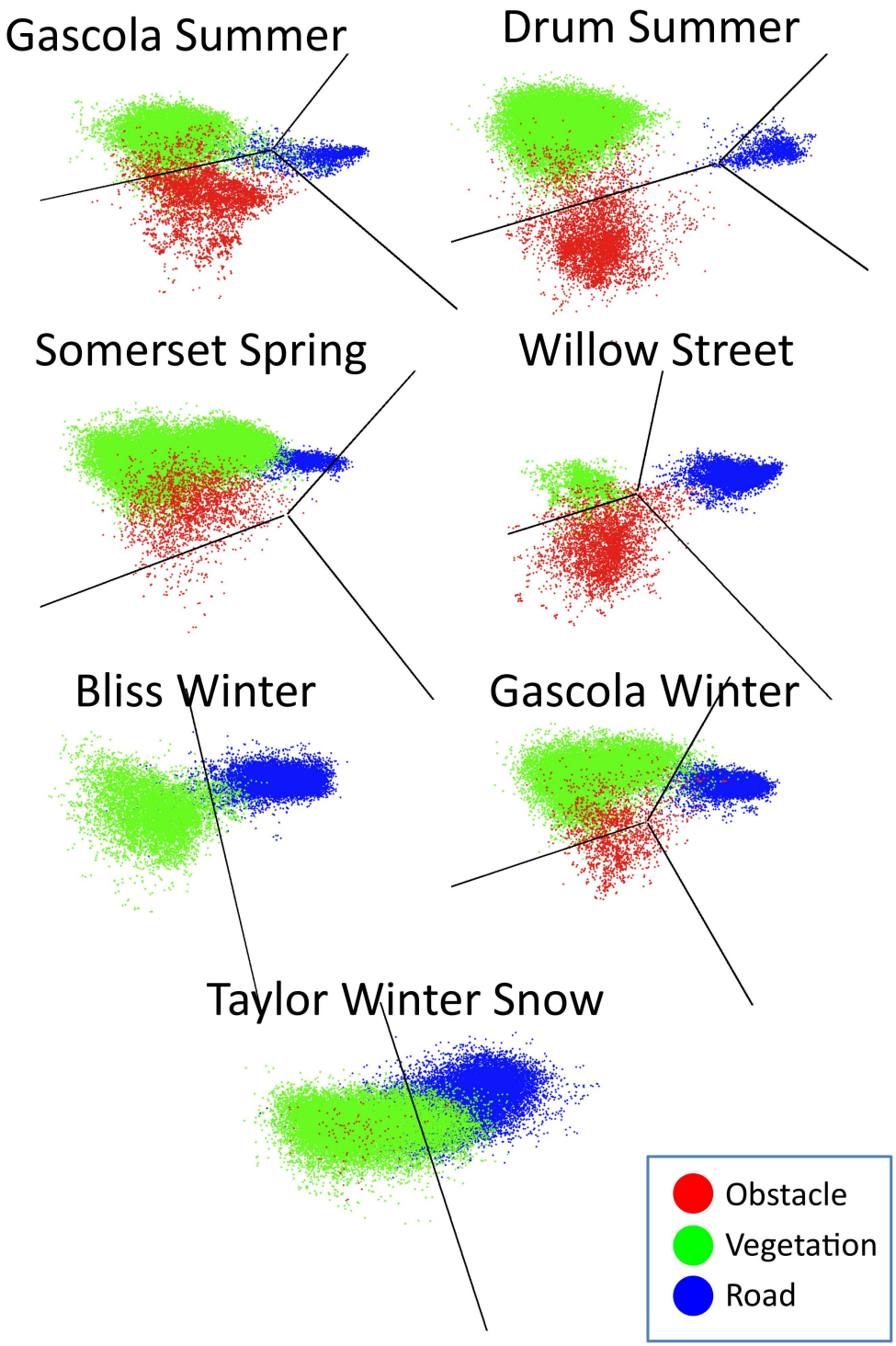


Figure 4.2: Different environments produce voxel class distributions, with substantially different optimal classification boundaries. Shown is the distribution of labeled data for 7 environments in the training set that had a good mix of "rigid" (road & obstacle) and "non-rigid" (vegetation) voxels. Labeled voxels are plotted as colored points in the most discriminative 2-D subspace (as computed by multi-class LDA across the entire training set). The optimal classification boundaries in each case are denoted by black lines.

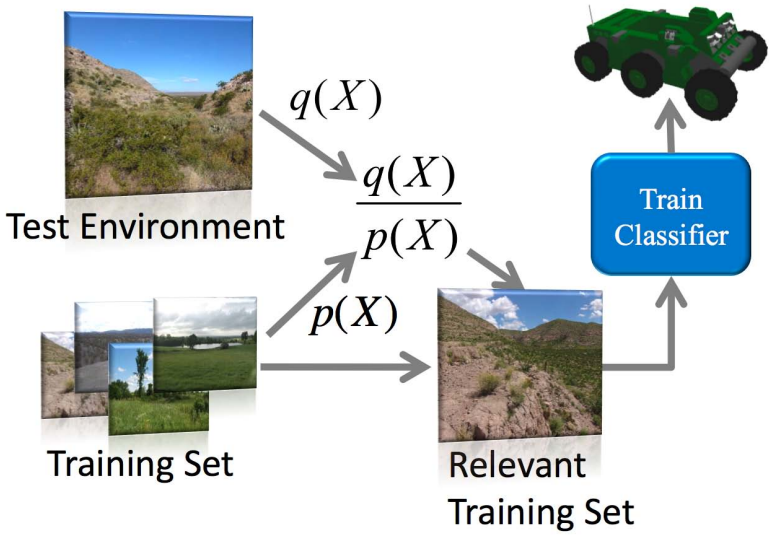


Figure 4.3: A training set drawn from a wide variety of different lighting, weather, and terrain conditions may be a poor match for a particular test environment. Importance sampling can be used to construct a training set that is more relevant to the test environment, by sampling training examples with a probability proportional to the ratio of unlabled data in the test environment q(X) and the training environment p(X).

$$E_{P^*(X,Y|\text{test})}[L(X,Y)] \approx \sum_{X,Y} P^*(X,Y|\text{test})L(X,Y) \tag{4.2}$$

$$= \sum_{X,Y} P^*(Y|X,\text{test})P^*(X|\text{test})L(X,Y)\frac{P^*(X,Y|\text{train})}{P^*(Y|X,\text{train})P^*(X|\text{train})}$$

$$= \sum_{X,Y} P^*(X,Y,|\text{train})L(X,Y)\frac{P^*(X|\text{test})}{P^*(X|\text{train})}$$

$$= E_{P^*(X,Y|\text{train})}\left[\frac{P^*(X|\text{test})}{P^*(X|\text{train})}L(X,Y)\right] \tag{4.3}$$

4.1.2 Assumption #2: constant labeling bias

A second assumption must be made, however, before (4.3) can be applied to mobile robot navigation. In practice the distribution of examples that are labeled is often different from the distribution of examples as some voxels (like large rocks) are more important for the vehicle to classify correctly and consequently have a higher weight in the labeled distribution through selective labeling or intentional class weighting. In the test environment, this "labeling bias" creates a distinction between $P^*(X|\text{test})$, the distribution of input vectors that are labeled, and the distribution $P(X|\text{test})$ of an unlabeled random sample. As only the random sample is available to the robot, the second assumption is that the ratio between the distributions of labeled and randomly-selected examples in

the training set is equal to the distribution between labeled and randomly-selected examples in the test set $\frac{P^*(X|\text{train})}{P(X|\text{train})} = \frac{P^*(X|\text{test})}{P(X|\text{test})}$. This assumption says that examples with the same input features should have the same importance in both the test and training distributions, and it depends on the input representation being discriminative and robust enough that important terrain types in one environment are not confused with less important terrain types in another environment. With this assumption, (4.3) can be rewritten as:

$$\begin{aligned} E_{P^*(X,Y|\text{test})}[L(X,Y)] &= E_{P^*(X,Y|\text{train})}\left[\frac{P^*(X|\text{test})}{P^*(X|\text{train})}\frac{P(X|\text{test})}{P^*(X|\text{test})}\frac{P^*(X|\text{train})}{P(X|\text{train})}L(X,Y)\right] \\ &= E_{P^*(X,Y|\text{train})}\left[\frac{P(X|\text{test})}{P(X|\text{train})}L(X,Y)\right]. \end{aligned} \tag{4.4}$$

The "importance weight", $w_i = \frac{P(\vec{x}_i|\text{test})}{P(\vec{x}_i|\text{train})}$, for each example in (4.4) can be calculated by estimating the density of unlabeled samples from the training and test environments. However, density estimation is difficult to do accurately in high dimensional spaces. To simplify the problem, Algorithm 1 first uses Linear Discriminants Analysis (LDA) on the full training set to compute a 2-D discriminative subspace for the three main classes of interest (Road, Vegetation, and Obstacle). A sample of data from each environment is projected into this 2-D subspace and used to fit a Gaussian Mixture Model (GMM). The GMMs–shown for each environment in Figure 4.4–are used to compute the probability densities required for the importance weights on the training examples.

Algorithm 1 Domain Adaptation from Unlabeled Data by Density Estimation

Input: Regularization parameter λ, labeled training set $\{X, Y\}$ and unlabeled samples $\{X\}$ from the test and training environments.

Output: Adapted classifier parameter vector $\vec{\theta}$.

1: Construct a low-dimensional subspace with LDA that preserves the separation between classes in the full labeled training set.

2: Project the unlabeled samples into the low-dimensional subspace.

3: Estimate $P(X|\text{test})$ and $P(X|\text{train})$ by fitting Gaussian Mixture Models (GMMs) to the unlabeled samples from the test and training environments.

4: Weight each example i in the labeled training set by $w_i = \frac{P(X_i|\text{test})+\lambda}{P(X_i|\text{train})+\lambda}$.

5: Return the parameter vector $\vec{\theta}$ which minimizes the total classification errors on the weighted training set:

$$\vec{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} w_i L(f(\theta, X_i), y_i)$$

4.2 Training Classifiers Instead of Estimating Densities

Implementing Algorithm (4.4) requires the estimation of two high-dimensional distributions, in order to compute the scalar weight for each data point. This is an unnecessarily difficult estimation

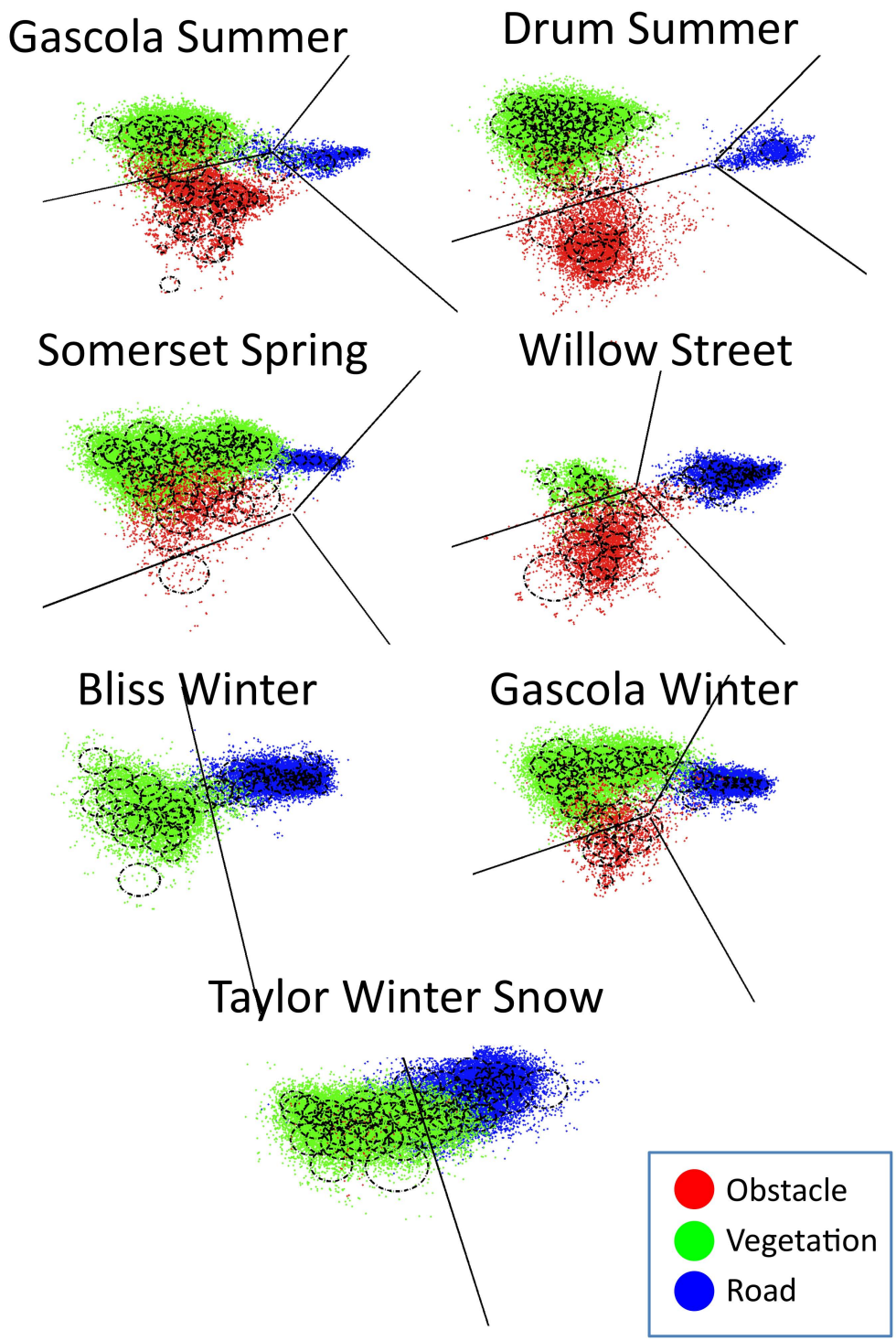


Figure 4.4: Algorithm 1 re-weights the training data by first fitting a Gaussian Mixture Model (GMM) to unlabeled samples from the test and training environments. Shown is the GMM for each environment, superimposed on the labeled data from Figure 4.2. The dashed circles correspond to the standard deviation of each spherical Gaussian in the GMM.

Algorithm 2 Domain Adaptation from Unlabeled Data by Classification

Input: Labeled training set $\{X, Y\}$ and unlabeled samples $\{X\}$ from the test and training environments.

Output: Adapted classifier parameter vector $\vec{\theta}$.

1: Train a logistic regression classifier to predict if an unlabeled feature vector X_i was drawn from the test or training environments.

2: Weight each example i in the labeled training set by $w_i = \frac{1}{P(\text{train}|X)} - 1$.

3: Return the parameter vector $\vec{\theta}$ which minimizes the total classification errors on the weighted training set:

$$\vec{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} w_i L(f(\theta, \vec{x}_i), y_i)$$

problem and an alternative approach introduced in [76] instead estimates a single conditional distribution, $p(\text{train}|X)$, with a probabilistic classifier such as logistic regression. Applying Bayes rule to the ratio of the probability of X under the training set, $p(X|\text{train})$, divided by the probability of X under the test set, $p(X|\text{test})$, produces:

$$\begin{aligned} \frac{p(X|\text{test})}{p(X|\text{train})} &= \frac{p(\text{test}|X)p(X)}{p(\text{test})} \frac{p(\text{train})}{p(\text{train}|X)p(X)} \\ &\propto \frac{P(\text{test}|X)}{p(\text{train}|X)} = \frac{1}{P(\text{train}|X)} - 1 \end{aligned} \tag{4.5}$$

Hence instead of estimating the probability density function of feature vectors in the training and testing sets independently, we can train a logistic regression classifier to predict if a particular feature vector came from the test or the training set. This classifier-based method for automatically adapting to a new unlabeled environment is given in Algorithm 2, and empirically produces better results on voxel classification than Algorithm 1.

4.3 Experiments

The GMM-based and classifier based domain adaptation algorithms were compared on a voxel classification task that forms a key component of Crusher's perception system. In this task features computed from camera and laser data of a local region of 3-D space (voxel) are classified into a rigid or a non-rigid terrain type. For certain parts of the system the rigid terrain type is subdivided into "road" voxels believed to be the ground surface and "obstacle" voxels believed to be above the ground surface. During the course of the UPI program, a large dataset of labeled voxels was collected from a variety of different terrain and seasonal conditions. From this dataset seven environments were defined that each had a mix of labeled rigid and non-rigid voxels. For each adaptation experiments, one of the seven environments was used as a test set and the other N-1 environments[2] were used to form a training set. For computational efficiency the test environment was

[2]Labeled data not in a balanced environment was also used in the training set.

randomly subsampled to 40,000 voxels and the training environment was subsampled to 360,000 voxels. Equally sized unlabeled data samples were also collected by randomly sampling from the sensor logs that the labeled samples were drawn from.

As accurate GMM estimation is difficult in high-dimensional spaces, for Algorithm 1 the 43-dimensional voxel feature space was projected to the 2-D subspace that best separated the Obstacle, Vegetation, and Road classes, as computed by MDA. The number of clusters in each GMM was fixed at 50, and a regularization parameter λ on the weights was chosen by leave one out cross-validation (Figure 4.5). The classifier based method, Algorithm 2 does not require dimensionality reduction and was able to use the full 43-dimensional input space. For this algorithm, the only free parameter–the regularization constant of the classifier–was chosen to be 0.01 by cross-validation (Figure 4.5). In both cases, relaxing the regularization boosted performance initially as it allowed the algorithms to be more aggressive in adapting the training set to match the observed test distribution. However, after a point the reduction in the sample size in the weighted training set allowed the classifier to over-fit, and hurt classification performance.

4.3.1 Voxel Classification Results

The classifier re-weighting technique (Algorithm 2) provided a performance boost for most of the environments tested. The results on each of the seven environments are shown in Table 4.2, and graphically in Figure 4.6. In all environments except the "Willow Street" test site, adaptation with the classifier-based method improved performance. Estimating the probability densities with Gaussian Mixture Models (GMMs) was not as effective on average, and fitting the GMMs proved to be more computationally intensive, and less beneficial than training logistic regression classifiers to estimate the data weights.

As the goal of domain adaptation is to approximate the test set with a weighted version of the training set, Table 4.2 also lists the "optimal" classification results of directly training on the test set. These results were included to show that due to the limited capacity of the linear classifier used for these experiments and the significant amount of label noise that exists in the training set (due to effects like vehicle pose error and labeling mistakes), it is not possible to achieve 100% classification accuracy even by training directly on the test set. Over all environments, the Logistic Regression (LR) classifier based algorithm achieved an average of 20% of the gap between the training set and the optimal results. The GMM based algorithm improved on some environments but was harmful on others.

Sampling high-probability training set images before and after adaptation provides insight into the operation of the algorithms. Figure 4.7 shows images randomly sampled form a test environment, and images randomly sampled from the re-weighted training set. One of the features of this environment is the presence of shipping containers, and images randomly sampled from the re-weighted training set reveal that after re-weighting, images from a different location that also contained shipping containers were weighted heavily. Also, examples from logs with lush vegetation were down-weighted in favor of examples from winter environments with dead vegetation.

The validity of the constant labeling bias assumption, $\frac{P^*(X|\text{train})}{P(X|\text{train})} = \frac{P^*(X|\text{test})}{P(X|\text{test})}$, can be tested by comparing the results of using the unlabeled data samples to reweight the training set (Figure 4.6) with results from cheating and using the labeled samples for re-weighting. The difference between using the unlabeled and labeled samples on each environment with the domain adaptation algorithms

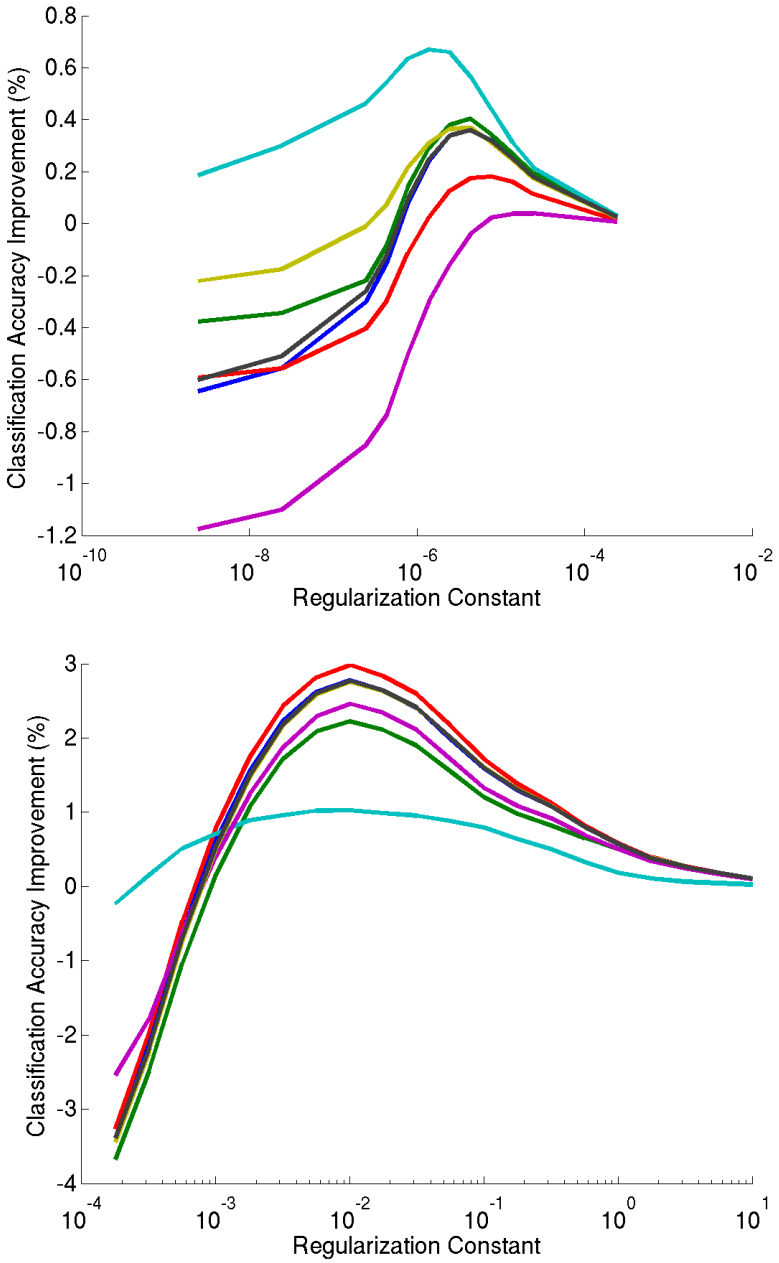


Figure 4.5: Free parameters for Algorithms 1 & 2 were chosen by leave-one-out cross validation. Each line shows the performance of a different fold as the regularization parameter is changed. Top: in Algorithm 1 regularization was implemented as a distribution smoothing parameter. Bottom: the performane of Algorithm 2 with repect to the regularization constant of the logistic regression classifier used to compute the importance weights.

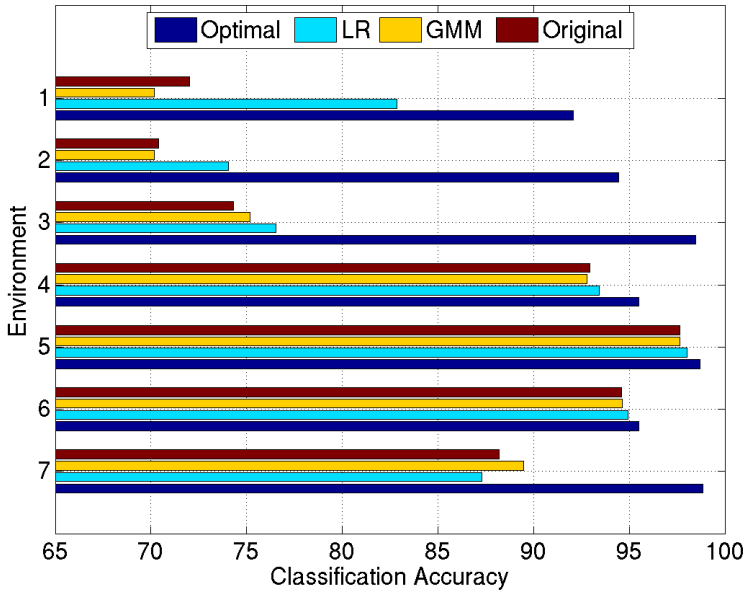


Figure 4.6: Performance on most environments benefitted from re-weighting. The top bar in each group is the performance of the classifier trained on the original training set and tested on this environment. The middle bars shows the performance after adaptation with the GMM-based Algorithm 1 and the Logistic Regression (LR) classifier approach of Algorithm 2. Finally, the bottom bar shows the "optimal" performance possible if the classifier were trained on the actual test set.

Figure 4.7: Randomly sampled images from a test environment (top) show dead vegetation and shipping containers. Sampled images from the reweighted training set (bottom) are more similar to the test environment than the original training set shown in Fugure 4.1.

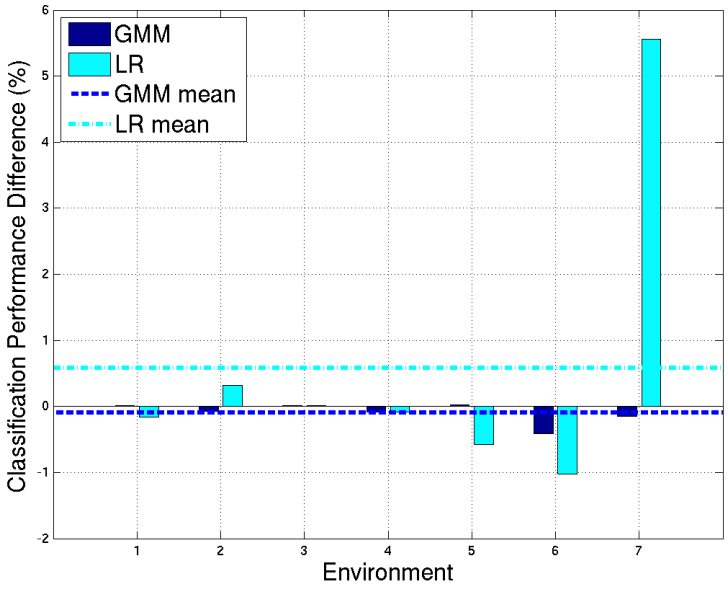


Figure 4.8: The assumption that the labeling bias is constant is hard to enforce in practice and is violated in this data set, as shown by the bar chart above. Each bas shows the difference in classification performance from estimating the importance weights using using unlabeled data vs only using labeled data. If the labeling bais assumption were perfectly true, there would be no performance difference. However, on this data set the positive and negative effects of labeling bias roughly cancel.

Environment	Original	GMM	LR	Optimal
Taylor Winter Snow	72.0%	69.5%	82.9%	92.1%
Gascola Summer	70.4%	70.2%	74.0%	94.5%
Bliss Winter	74.3%	75.2%	76.5%	98.5%
Sommerset Spring	92.9%	92.9%	93.4%	95.5%
Drum summer	97.6%	97.7%	98.0%	98.7%
Gascola Winter	94.6%	94.6%	94.9%	95.5%
Willow Street	88.2%	89.1%	87.3%	98.9%
Mean	84.3%	84.2%	86.7%	96.2%

Table 4.2: Adaptation Performance on each environment. Environments which are poorly matched to the full training set show the greatest gains. The left column is the test set performance of classifiers trained on the full training set. The logistic regression (LR) method presented in Algorithm 2 performed better on most environments than the GMM-based density estimation in Algorithm 1. In all cases except the Willow Street environment, adaptation from unlabeled data using the classifier-based method improved classification performance. The right column shows the result of training directly on the test set for each environment, and provides an upper bound on the performance of any domain adaptation algorithm. The regularization parameter was selected by leave-one-out cross validation.

is shown in Figure 4.8. Perhaps surprisingly there is no consistent gain from using the labeled sample, and both algorithms had a mix of positive and negative effects. This result may also reflect the difficulty of maintaining a constant labeling bias across environments. The dataset used for this experiment was labeled by multiple people as specific performance deficiencies in the Crusher system were identified, and labeling in each sensor log typically focused on specific terrain types of interest. Because of this labeling process, it is likely that the second assumption was often violated. However, the effect of this assumption was positive as often as it was negative, and on average it seems not to hurt performance.

Interestingly, neither of the domain adaptation algorithms necessarily makes the marginal class distribution after re-weighting closer to the label class distribution in the test set. Figure 4.9 compares the probability of the rigid class in the original training set, re-weighted training set, and test set for each environment. In this experiment, the largest gains occurred when the adaptation process made the marginal class distribution of the adapted training set less similar to the marginal class distribution of the test set.

4.3.2 System-level Improvement

A preliminary experiment was also conducted on the impact of this algorithm on final-system performance, which showed promising performance improvements. The robot traversed a 1.5 Km course twice, once with classifiers trained from the original labeled training set, and again with classifiers trained from an adapted training set. The adaptation algorithm made the robot more willing to call dead November vegetation non-rigid, and led to faster speeds and a more efficient route

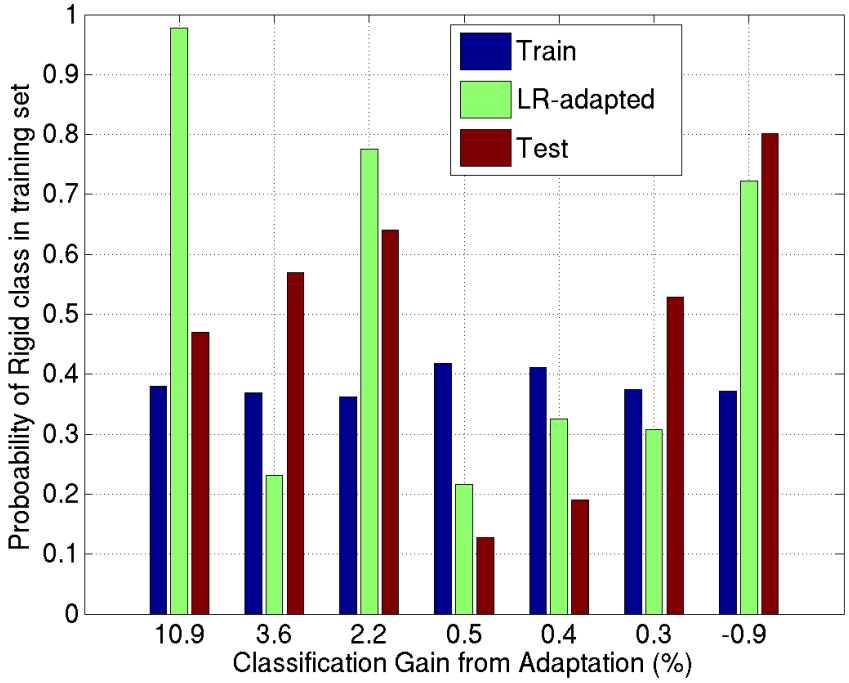


Figure 4.9: Domain adaptation does not necessarily make the class distribution in the re-weighted training set closer to the class distribution in the test set. For several environments adaptation makes the class distribution significantly less similar while still improving classification performance.

which shaved 13% off of the total run. However due to limited robot availability further system-level experiments were not possible.

Chapter 5

Sparse Coding In Modular Systems

Recent research [4, 3] shows that pre-training early modules in a deep network to model their inputs can dramatically improve the generalization performance of the trained network. Sparse coding is one approach to input modeling that demonstrates particular promise for learning good features for classification [21]. However, the utility of sparse coding was limited by the fact that sparse coding modules were non-differentiable, implying they would be difficult to optimize for a particular prediction task by backpropagation. This chapter addresses this concern by deriving a differentiable formulation of sparse coding [77] that can be used with backpropagation to specialize the parameters of the sparse coding module for a prediction task.

5.1 Sparse Coding

Many learning problems can be phrased as *matrix factorization* or *coding* problems, where the goal is to represent a collection of input signals in terms of a set of factors that are shared across all input examples, and a set of factors that are specific to each input example. For instance, if the input examples are arranged as columns of a matrix $X \in \Re^{m \times n}$, the goal of the matrix factorization problem is to find matrices $B \in \Re^{m \times d}$ and $W \in \Re^{d \times n}$ such that:

$$X \approx f(BW). \tag{5.1}$$

The basis matrix B provides a "code", a dictionary of elements that is shared across all examples. Each column of the coefficient matrix W contains coefficients to reconstruct the corresponding column of X as a weighted combination of the columns of B. Here $f(z)$ is an optional element-wise link function. Various approaches to sparse approximation, sparse coding, dimensionality reduction, independent component analysis, clustering and even boosting can be written as variants of (5.1). The next two chapters will examine *optimization modules*, which solve the matrix factorization problem by assuming probability models $P(W)$, $P(B)$, $P(X|BW)$, and using optimization to infer the Maximum A-Posteriori (MAP) values $\hat{W}$ and $\hat{B}$ which maximizes the joint probability:

$$[\hat{W}, \hat{B}] = \arg\max_{W,B} P(X|B, W)P(B)P(W) \tag{5.2}$$

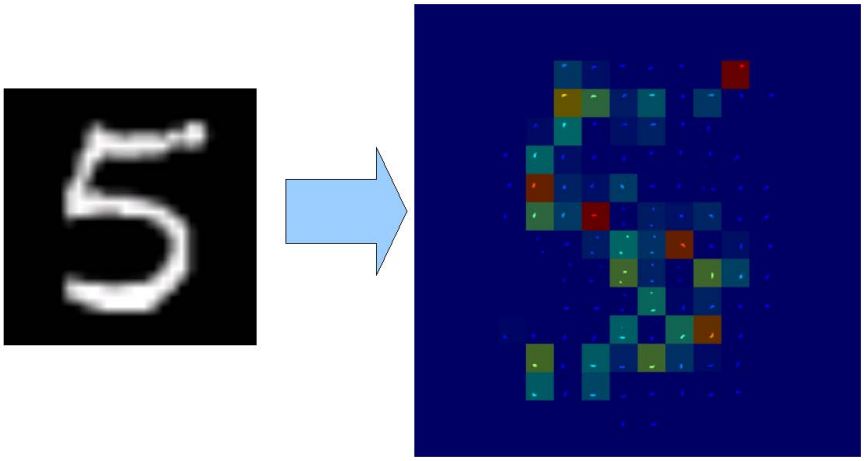


Figure 5.1: An example basis learned by sparse coding decomposes an image of a handwritten digit (left) as a weighted sum of basis vectors (right). The coloring of the basis vectors indicates the magnitude of their weight, with blue vectors contributing almost nothing and red vectors having a large weight.

Certain choices for the prior over coefficients, $P(W)$, are known to favor "sparse" solutions where many of the entries of $\hat{W}$ are exactly zero. Such priors, especially L_1 regularization, have gained attention because of their usefulness in ill-posed engineering problems [78], elucidating neuro-biological phenomena, [79, 80],face recognition [81], and semi-supervised and transfer learning [21, 77, 82]. An example of sparse coding an image of a handwritten digit is given in Figure 5.1.

5.1.1 Notation

Uppercase letters, X, denote matrices and lowercase letters, x, denote vectors. For matrices, superscripts and subscripts denote rows and columns respectively. X_j is the j^{th} column of X, X^i is the i^{th} row of X, and X^i_j is the element in the i^{th} row and j^{th} column. Elements of vectors are indicated by subscripts, x_j. X^T is the transpose of matrix X. The Fenchel conjugate of a function f is denoted by f^*. The notation $(x)_+$ means the larger of 0 or x.

5.1.2 Generative Model

From a probabilistic viewpoint, sparse coding fits a generative model (5.3), which maximizes the likelihood of a set of unlabeled data given by the matrix X, where each column of X is an unlabeled input example.

$$\begin{aligned} P(X) &= \int_B \int_W P(X|BW)P(W)P(B)dWdB \\ &= \int_B dP(B) \int_W \prod_i P(X_i|BW_i)dP(W_i) \end{aligned} \tag{5.3}$$

Applying the Maximum A Posteriori (MAP) approximation replaces the integration over W and B in (5.3) with its maximum value $P(X|\hat{B}\hat{W})P(\hat{W})P(\hat{B})$, where the values of the latent variables at the maximum, $\hat{W}$ and $\hat{B}$, are referred to as the MAP estimates. Finding $\hat{W}$ given B is an approximation problem; solving for $\hat{W}$ and $\hat{B}$ simultaneously over a set of examples is a coding problem. The coding problem is solved in [83] by minimizing the negative log of the generative model (5.4), subject to the constraint that the columns of the basis matrix have unit length:

$$\begin{aligned} \hat{W}, \hat{B} &= \arg\min_{W,B} L(X, f(BW)) + \lambda\Phi(W) \\ &s.t. \quad \|B_i\|_2 = 1, \ \forall i. \end{aligned} \tag{5.4}$$

The loss function, $L(X, f(BW))$, can be interpreted as the negative log of the conditional probability $P(X|BW)$. $f(r)$ is an optional *transfer function* that applies an operation to each element of the product BW. We will focus on the common case where the loss function is the square of the Frobenius norm, $L(X, BW) = \|BW - X\|_F^2$, of the difference between the input matrix X and its reconstruction with the linear transfer function $f(BW) = BW$. This choice of matching loss and transfer functions assumes that the input X is corrupted with additive Gaussian noise. The regularization constant λ is proportional to the expected variance of that noise. More generally, the distribution of X can be modeled by an exponential family distribution. Every exponential family distribution defines a Bregman divergence which serves as a matching loss function for estimating the parameters of the distribution[1].

Similarly, the regularization function $\Phi(W)$ may be seen as the negative log of the prior distribution on the coefficients. The Laplacian prior distribution, which corresponds to regularization with an L_1-norm[2], has received much interest recently because it tends to prefer $\hat{W}$ which are "sparse" in that they have a small number of non-zero elements, even when the basis matrix has infinitely many columns. This preference has been shown to be useful for applications such as prediction [21, 77, 82] and denoising of images and video [85]. Substituting this choice of regularization and loss functions produces the optimization problem:

$$\begin{aligned} \hat{W}, \hat{B} &= \arg\min_{W,B} \|BW - X\|_F^2 + \lambda\|W\|_1 \\ &s.t. \quad \|B_i\|_2 = 1, \ \forall i. \end{aligned} \tag{5.5}$$

5.2 Differentiable Sparse Coding

Past work [21] shows that sparse coding can learn useful features for classification from unlabeled data on a wide variety of problems. However, in general there is no guarantee that a basis matrix B, which can be used to closely approximate a set of unlabeled input data $X \approx f(BW)$, will produce coefficients W that are also useful for prediction tasks $g(W) \approx Y$. As discussed previously, an

[1] The maximum likelihood parameter estimate for any regular exponential family distribution can be found by minimizing the corresponding Bregman divergence for that family, and every Bregman divergence has a matching transfer function which leads to a convex minimization problem [84]. That matching transfer function is the gradient $\nabla\phi$ of the function ϕ which is associated with the Bregman divergence $D_\phi(x\|y) = \phi(x) - \phi(y) - \langle x - y, \nabla\phi(y)\rangle$.

[2] The L_p norm of a vector x is: $\|x\|_p = \left(\sum_i |x|_i^p\right)^{1/p}$, for $p \geq 1$

effective strategy for deep belief networks and stacked autoassociators is to pre-train each module to reconstruct its input, and then train the whole network by gradient descent (backpropagation) on the loss function for a target task. However, priors such as L_1 which produced sparse W, also can make W discontinuous with respect to small changes in X and B; an arbitrarily small change in input can lead to the selection of a completely different set of latent weights. This discontinuity violates a central assumption of gradient descent: that a local minima can be reached by small incremental gradient steps.

This work argues for a non-traditional definition of sparsity that is better suited for using the optimization module as part of a larger learning system. It also gives examples of priors that satisfy this pseudo-sparsity, and shows that MAP estimates computed under these priors are more useful as input to a wide range of classifiers. Most importantly, it demonstrates that implicit differentiation can be used with a large class of pseudo-sparse priors to differentiate the latent weights W with respect to the input X and the basis B. Differentiability allows classification error gradients to be used to update the sparse coding basis, and produces a powerful new semi-supervised learning tool. An alternative approach pursued in [82] added classification loss directly to the sparse coding objective function, which creates optimization difficulties but produced promising results.

5.2.1 Approximately Sparse Coding

A vector is commonly called "sparse" if many elements are exactly zero. The entropy [86, 87], and L_p^p, $p \leq 1$ [79, 80, 21] regularization functions[3] promote this form of sparsity, and all of them have shown the ability to learn bases containing interesting structure from unlabeled data. However, of these only L_1 leads to an efficient, convex procedure for inference, and even this prior does not produce differentiable MAP estimates.

If the latent weights $\hat{W}$ are to be used as input to a classifier, a better definition of "sparsity" is that almost all of the deviation between each column w of W and a constant vector p occurs in a small number of elements. One regularization function that produces this form of pseudo-sparsity is the KL-divergence $KL(w\|p)$. This regularization function has long been used for approximation problems in Geophysics, Crystallography, Astronomy, and Physics, where it is commonly referred to as Maximum Entropy on the Mean (MEM) [88], and has been shown in the online setting to compete with low L_1-norm solutions in terms of regret [30, 28].

L_1 regularization provides sparse solutions because its convex dual is the max function, meaning only the most useful basis vectors are selected to participate in the reconstruction. A differentiable approximation to $\max_i x_i$ is a sum of exponentials, $\sum_i e_i^x$, whose dual is the KL-divergence (5.6). Regularization with KL has proven useful in online learning, where it is the implicit prior of the exponentiated gradient descent (EGD) algorithm (Section 2.1.4). EGD has been shown to be "sparse" in the sense that it can select a few relevant features to use for a prediction task from many irrelevant ones.

The form of KL used here is the full Bregman divergence of the negative entropy function[4]:

$$\Phi(w) = \sum_i w_i \log \frac{w_i}{p_i} - w_i + p_i. \tag{5.6}$$

[3] L_p^p corresponds to the negative log of a generalized gaussian prior.

[4] $-H(x) = \sum_i x_i \log(x_i)$

Often KL is used to compute distances between probability distributions, and for this case $\|\hat{w}\|_1 = \|p\|_1 = 1$ and (5.6) reduces to the standard form. This is an inconvenient assumption for sparse coding, however, so the full unnormalized KL is used instead.

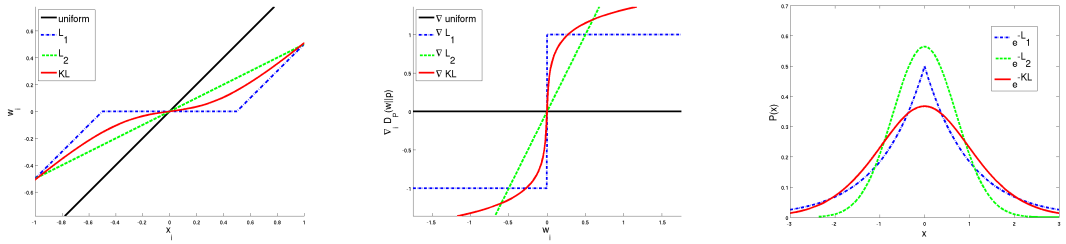
A reasonable choice for the prior vector p is a uniform vector whose L_1 magnitude equals the expected L_1 magnitude of w. p has an analogous effect to the q parameter in L_q-norm regularization. $p \to 0$ approximates L_1 and $p \to \infty$ approximates L_2. Changing p affects the magnitude of the KL term, so the regularization constant λ in (5.4) must be adjusted to balance the loss term in the sparse coding objective function (small values of p require small values of λ).

5.2.2 Properties of the Unnormalized KL-divergence Prior

Many of the important properties of a regularization function can be understood by looking at the behavior of its partial derivatives, since at $\hat{w}$ the (sub)gradient of the regularization function cancels the gradient of the loss:

$$\nabla\Phi(\hat{w}) = -\frac{1}{\lambda}\nabla L\left(x, f(B\hat{w})\right). \tag{5.7}$$

In the case where B is the identity matrix and $L(x, f(Bw)) = \frac{1}{2}\|x - Bw\|_2^2$, each element $\hat{w}_i$ can be computed independently as $\hat{w}_i = x_i - \nabla_i\Phi(\hat{w})$. Figure 5.2A plots $\hat{w}_i$ against x_i for various priors. The uniform prior has no effect and produces the line $\hat{w}_i = x_i$. An L_2 (gaussian) prior changes the slope of the line to $\hat{w}_i = (1 - \lambda)x_i$, but does not change the sparsity of $\hat{w}$, as all elements are scaled equally. An L_1 prior however, does change the sparsity because its gradient is discontinuous at zero which forces $\hat{w}_i = 0$ while $|x_i| \leq \lambda$.



A B C

Figure 5.2: For the identity basis, the gradient of the log prior on $\hat{w}$ (B) determines an offset between $\hat{w}_i$ and x_i (A). The probability density functions obtained by exponentiating the L_1, L_2, and KL regularization functions are shown in (C). Note that KL regularization is shown on an expanded basis set that adds the negation of each basis vector to allow negative weights.

KL-divergence regularization does not allow negative weights, but expanding the basis by adding the negation of each basis vector $\widetilde{B} = [-B\ B]$ simulates the effects of negative weights with only positive ones. In this case the derivative of the KL-divergence prior corresponds to the arcsinh (w_i/p_i) function pictured in Figure 5.2B. arcsinh grows quickly for small weights, which causes sparsity similar to L_1. Crucially though, it grows slowly for large weights while still reaching ∞. This property causes $\hat{w}$ to be differentiable and stable to small changes in B and x, by allowing $\hat{w}$ to contain large weights while still ensuring that similar columns of B will have similar activations in $\hat{w}$. Be-

cause its derivative is flat apart from $w = 0$, L_1 regularization can produce discontinuous solutions for $\hat{w}$ if the columns of B are not orthogonal[5].

Relationship between KL-divergence and arcsinh

The MAP estimate $\widetilde{w}$ obtained for KL regularization on the basis $\widetilde{B} = [-B\ B]$ is related through the function $\hat{w} = \widetilde{w}^+ - \widetilde{w}^-$ to the MAP estimate $\hat{w}$ produced by using the regularization function $\sum_i w_i \text{arcsinh}\left(\frac{w_i}{2p_i}\right) - \sqrt{w_i^2 + 4p_i^2}$ (shown in figure 5.2) on the basis B.

Define $\widetilde{B} = [-B\ B]$, and compute the MAP estimate $\widetilde{w}$ by minimizing (5.8).

$$\widetilde{w} = \arg\min_w L\left(x, f(\widetilde{B}w)\right) + \lambda \text{KL}(w\|p) \tag{5.8}$$

At the minimum $\widetilde{w}$, the gradients of both terms in (5.8) cancel each other (5.9).

$$-\frac{1}{\lambda}\frac{\partial}{\partial \widetilde{w}} L\left(x, f(\widetilde{B}\widetilde{w})\right) = \frac{\partial}{\partial \widetilde{w}} \text{KL}(\widetilde{w}\|p) \tag{5.9}$$

Divide the elements of $\widetilde{w}$ into two groups $\widetilde{w} = \begin{bmatrix} \widetilde{w}^- \\ \widetilde{w}^+ \end{bmatrix}$, so that $\widetilde{B}\widetilde{w} = B(\widetilde{w}^+ - \widetilde{w}^-) = B\hat{w}$, where $\hat{w}$ is defined to be $\hat{w} = \widetilde{w}^+ - \widetilde{w}^-$. Notice that $\frac{\partial(\widetilde{B}\widetilde{w})}{\partial \widetilde{w}} = \begin{bmatrix} -\frac{\partial B\hat{w}}{\partial w} & \frac{\partial B\hat{w}}{\partial w} \end{bmatrix}$. Substituting $B\hat{w}$ into (5.9) produces:

$$-\frac{1}{\lambda}\begin{bmatrix} -\frac{\partial}{\partial \hat{w}} L(x, f(B\hat{w})) \\ \frac{\partial}{\partial \hat{w}} L(x, f(B\hat{w})) \end{bmatrix} = \begin{bmatrix} \log \frac{\widetilde{w}^-}{p} \\ \log \frac{\widetilde{w}^+}{p} \end{bmatrix}. \tag{5.10}$$

Solve (5.10) for $\widetilde{w}_i^+$ and $\widetilde{w}_i^-$:

$$\begin{aligned} \widetilde{w}^+ &= p e^{-\frac{1}{\lambda}\frac{\partial}{\partial w} L(x, f(B\widetilde{w}))} \\ \widetilde{w}^- &= p e^{\frac{1}{\lambda}\frac{\partial}{\partial w} L(x, f(B\widetilde{w}))}. \end{aligned} \tag{5.11}$$

Rewrite $\hat{w} = \widetilde{w}^+ - \widetilde{w}^-$ using (5.11) and the hyperbolic sin function $\sinh(x) = 1/2(e^x - e^{-x})$.

$$\begin{aligned} \hat{w} &= p e^{-\frac{1}{\lambda}\frac{\partial}{\partial w} L(x, f(B\hat{w}))} - p e^{\frac{1}{\lambda}\frac{\partial}{\partial w} L(x, f(B\hat{w}))} \\ &= 2p \sinh\left(-\frac{1}{\lambda}\frac{\partial}{\partial w} L(x, f(B\hat{w}))\right) \end{aligned} \tag{5.12}$$

Rearranging produces (5.13), which is the derivative of (5.14) at its MAP estimate. Hence $\hat{w}$ computed by KL-regularization is also the MAP estimate of (5.14).

$$-\frac{1}{\lambda}\frac{\partial}{\partial \hat{w}} L(x, f(B\hat{w})) = \text{arcsinh}\left(\frac{\hat{w}}{2p}\right) \tag{5.13}$$

$$\hat{w} = \arg\min_w L(x, f(Bw)) + \lambda \sum_i w_i \text{arcsinh}\left(\frac{w_i}{2p_i}\right) - \sqrt{w_i^2 + 4p_i^2} \tag{5.14}$$

[5]For an extreme, but illustrative, example consider the case where B contains two identical basis vectors. Then there is no longer even a unique $\hat{w}$.

5.2.3 Transforming a Generative Model Into A Discriminative One

Sparse coding builds a generative model for a set of unlabeled data by learning the basis matrix B. Our hope is that the MAP estimate of basis coefficients $\hat{w}$ produced for each input vector x will be useful for predicting a response y associated with x. However, the sparse coding objective function only penalizes input reconstruction errors, and does not necessarily make $\hat{w}$ useful as input for any particular task. Fortunately, since priors such as KL-divergence regularization produce solutions that are smooth with respect to small changes in B and x, B can be modified through backpropagation to make $\hat{w}$ more useful for prediction.

The key to computing the derivatives required for backpropagation is noting that the gradient with respect to w of the optimization (5.4) at its minimum $\hat{w}$ can be written as a set of fixed point equations (5.7) where the gradients of the loss term equal the gradient of the regularization. If the regularization function $\Phi(w)$ is twice differentiable with respect to w, we can use *implicit differentiation* on (5.7) to compute the gradient of $\hat{w}$ with respect to B, and x [89]. For KL-regularization and the simple case of a linear transfer function with squared loss, each element of the gradient $\frac{\partial \hat{w}}{\partial B}$ is:

$$\frac{\partial \hat{w}}{\partial B_i^k} = -\left(B^T B + \text{diag}(\frac{\lambda}{\hat{w}})\right)^{-1} \left((B^k \hat{w}_i)^T + \vec{e}_i(f(B^k \hat{w}) - x_k)\right), \tag{5.15}$$

where $\vec{e}_i$ is a unit vector whose i^{th} element is 1. If the basis vectors (columns of B) are linearly independent, the matrix $\left(B^T B + \text{diag}(\frac{\lambda}{\hat{w}})\right)^{-1}$ will be positive definite, otherwise it is guaranteed to be at least positive semi-definite [90]. As positive semi-definite matrices do not have negative eigenvalues, modifying (5.15) by removing the multiplication by $\left(B^T B + \text{diag}(\frac{\lambda}{\hat{w}})\right)^{-1}$ will not be negatively correlated with the true gradient. In the experiments performed in Section 5.4, the matrix inverse was efficiently computed using the strategies discussed in Section 5.3.1, and it did not present a computational bottleneck. Note also that the ability to compute both $\frac{\partial \hat{w}}{\partial x}$ and $\frac{\partial \hat{w}}{\partial B}$ means that an optimization module can be used at any point in a generalized backpropagation network, not just for the first layer.

Derivative for matching loss/transfer function pairs

It is also possible to derive $\frac{\partial \hat{w}}{\partial B}$ for general pairs of matching reconstruction loss $L(x, r)$ and reconstruction transfer functions, $r = f(Bw)$, whose derivatives with respect to the coefficients assume a common form:

$$\frac{\partial L(x, r)}{\partial w} = B^T (r - x). \tag{5.16}$$

Examples of such pairs include the linear output function with squared loss, and normalized exponential reconstruction with KL loss. Differentiating (5.16) with respect to B_i^k produces the vector equation:

$$\frac{\partial}{\partial B_i^k}\left(\frac{\partial L(x, r)}{\partial w}\right) = B^T \left(\frac{\partial r}{\partial w}\frac{\partial w}{\partial B_i^k} + \frac{\partial r}{\partial B_i^k}\right) + \vec{e}_i(r_k - x_k). \tag{5.17}$$

Here $\vec{e}_i$ is a unit vector whose i^{th} element is 1. Similarly, the derivative of the gradient of the KL-divergence prior with respect to B_i^k is the product of a diagonal matrix and the column vector $\frac{\partial w}{\partial B_i^k}$

(5.18).

$$\frac{\partial}{\partial B_i^k}\left(\frac{\partial \Phi(w)}{\partial w}\right) = \text{diag}\left(\frac{1}{w}\right)\frac{\partial w}{\partial B_i^k} \tag{5.18}$$

At the MAP estimate $\hat{w}$, (5.17) equals negative (5.18), and solving for $\frac{\partial \hat{w}}{\partial B_i^k}$ we get (5.19).

$$\text{diag}(\frac{-\lambda}{\hat{w}})\frac{\partial \hat{w}}{\partial B_i^k} = B^T\left(\frac{\partial \hat{r}}{\partial \hat{w}}\frac{\partial \hat{w}}{\partial B_i^k} + \frac{\partial \hat{r}}{\partial B_i^k}\right) + \vec{e}_i(\hat{r}_k - x_k)$$

$$-\left(B^T\frac{\partial \hat{r}}{\partial \hat{w}} + \text{diag}(\frac{\lambda}{\hat{w}})\right)\frac{\partial \hat{w}}{\partial B_i^k} = B^T\frac{\partial \hat{r}}{\partial B_i^k} + \vec{e}_i(\hat{r}_k - x_k)$$

$$\frac{\partial \hat{w}}{\partial B_i^k} = -\left(B^T\frac{\partial \hat{r}}{\partial \hat{w}} + \text{diag}(\frac{\lambda}{\hat{w}})\right)^{-1}\left(B^T\frac{\partial \hat{r}}{\partial B_i^k} + \vec{e}_i(\hat{r}_k - x_k)\right) \tag{5.19}$$

This general form can be used with many loss/transfer function pairs by substituting the appropriate partial derivatives. Two common examples are listed in the table below.

Transfer Function	$r(B, w)$	$\frac{\partial r}{\partial w}$	$\frac{\partial r}{\partial B_i^k}$
Linear	Bw	B	$\vec{e}_k w_i$
Normalized Exponential	$\frac{e^{Bw}}{\sum_j e^{B^j w}}$	$(\text{diag}(r) - rr^T)B$	$(\vec{e}_k - r)w_i r_k$

Linear Transfer Function with squared loss:

$$\frac{\partial \hat{w}}{\partial B_i^k} = -\left(B^T B + \text{diag}(\frac{\lambda}{\hat{w}})\right)^{-1}\left((B^k \hat{w}_i)^T + \vec{e}_i(\hat{r}_k - x_k)\right) \tag{5.20}$$

Normalized exponential transfer function with KL loss:

$$\frac{\partial \hat{w}}{\partial B_i^k} = -\left(B^T(\text{diag}(\hat{r}) - \hat{r}\hat{r}^T)B + \text{diag}(\frac{1}{\eta \hat{w}})\right)^{-1}\left(B^T(\vec{e}_k - \hat{r})\hat{w}_i\hat{r}_k + \vec{e}_i(\hat{r}_k - x_k)\right) \tag{5.21}$$

5.3 Implementation

Computing $\hat{w}$ with KL-regularization, was done by minimizing (5.4) using exponentiated gradient descent (EGD) with backtracking until convergence. EGD uses the iterative update rule:

$$w_j^{t+1} = w_j^t e^{-\eta\frac{\partial \varrho}{\partial w_j}}, \tag{5.22}$$

which automatically enforces positivity constraints on the coefficient vector w, and is particularly efficient for optimization because it is the natural mirror descent rule for KL-regularization [28]. The parameter η is the step size. Instead of doing a full line search on each step in the direction of the gradient, the backtracking approach starts with a large step size, and evaluates the objective after the step. If the objective fails to improve, the step size is decreased by a half and the objective is recomputed. This process is repeated until a step is found which improves the objective. Here

$\mathfrak{L}$ refers to the sparse coding objective function (5.4). For matching loss/transfer function pairs the gradient of $\mathfrak{L}$ with respect to the coefficient for the j^{th} basis vector w_j is:

$$\frac{\partial \mathfrak{L}}{\partial w_j} = (f(Bw) - x)^T B_j + \lambda \log \frac{w_j}{p_j}. \tag{5.23}$$

This iterative update is run until the maximum gradient element is less than a threshold, which is estimated by periodically running a random set of examples to the limits of machine precision, and selecting the largest gradient threshold that produces $\hat{w}$ within ϵ of the exact solution. The η parameter is continuously updated to balance the number of successful steps and the number of backtracking steps[6]. Because L_1-regularization produces both positive and negative weights, to compare L_1 and KL regularization on the same basis we expand the basis used for KL by adding the negation of each basis vector, which is equivalent to allowing negative weights (Section 5.2.2).

During sparse coding the basis matrix B is optimized by the Stochastic Gradient Descent (SGD) update rule: $B_{t+1} = B_t - \eta \frac{\partial \mathfrak{L}}{\partial B^i_j}$. For matching loss/transfer function pairs $\frac{\partial \mathfrak{L}}{\partial B^i_j}$ is:

$$\frac{\partial \mathfrak{L}}{\partial B^i_j} = w_j(f(B^i w) - x_i). \tag{5.24}$$

Note that the update equation does not depend on the prior chosen for w. SGD implements an implicit L_2 regularizer and is suitable for online learning, however because the magnitude of w is explicitly penalized, the columns of B were constrained to have unit L_2 norm to prevent the trivial solution of infinitely large B and infinitely small w. The step size was adjusted for the magnitude of $\hat{w}$ in each application, and was then decayed over time as $\eta \propto 1/\sqrt{t}$. The same SGD procedure was also used to optimize B through backpropagation, as explained in Section 5.2.3.

5.3.1 Computing the Basis Gradient with Small Weights

When some of the coefficients are close to zero, the matrix inverse $\left(B^T B + \text{diag}(\frac{\lambda}{w})\right)^{-1}$ is ill-conditioned. Fortunately, the structure of the matrix allows us to use the matrix inversion lemma to compute the inverse exactly (5.25). This approach can also lead to computational efficiency gains if the number of basis vectors, d, is greater than the number of input dimensions, m, as it requires the inversion of an mxm matrix instead of a dxd matrix:

$$\left(B^T \frac{\partial r}{\partial w} + \text{diag}\left(\frac{w}{\lambda}\right)\right)^{-1} = \text{diag}\left(\frac{w}{\lambda}\right)\left[I - \frac{w}{\lambda} B^T \left(I + \frac{\partial r}{\partial w}\text{diag}\left(\frac{w}{\lambda}\right) B^T\right)^{-1} \frac{\partial r}{\partial w}\text{diag}\left(\frac{w}{\lambda}\right)\right]. \tag{5.25}$$

Alternatively, if the number of significantly non-zero coefficients is small relative to the number of input dimensions, the Schur complement can be used to provide a fast approximation to the matrix inverse because the sub-matrix corresponding to basis vectors with very small weights is almost diagonal. First, multiply by a permutation matrix P to rearrange the ordering of the weight vector

[6] In our experiments, if the ratio of backtracking steps to total steps was more than 0.6, η was decreased by 10%. Similarly η was increased by 10% if the ratio fell below 0.3.

and the columns of B to form a sub-matrix C containing only weights smaller than a threshold, producing the block diagonal structure :

$$P\left(B^T \frac{\partial r}{\partial w} + \text{diag}(\frac{\lambda}{w})\right) = \begin{bmatrix} A & F \\ F^T & C \end{bmatrix} \tag{5.26}$$

Note that the matrix C corresponding to the small weights is essentially a diagonal matrix with very large components on the diagonal, and its inverse C^{-1} will be close to 0. Next use the Schur complement (5.27) to compute the inverse of the block matrix (5.28) ([29] page 650):

$$S = C - F^T A^{-1} F \tag{5.27}$$

$$\begin{bmatrix} A & F \\ F^T & C \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1} F S^{-1} F^T A^{-1} & -A^{-1} F S^{-1} \\ -S^{-1} F^T A^{-1} & S^{-1} \end{bmatrix}. \tag{5.28}$$

Because the diagonal elements of C are much greater than the elements of $F^T A^{-1} F$, the Schur complement S is approximately diagonal with very large elements, and it's inverse is essentially zero, leading to the approximation (5.29).

$$\begin{bmatrix} A & F \\ F^T & C \end{bmatrix}^{-1} \approx \begin{bmatrix} A^{-1} & 0 \\ 0 & 0 \end{bmatrix} \tag{5.29}$$

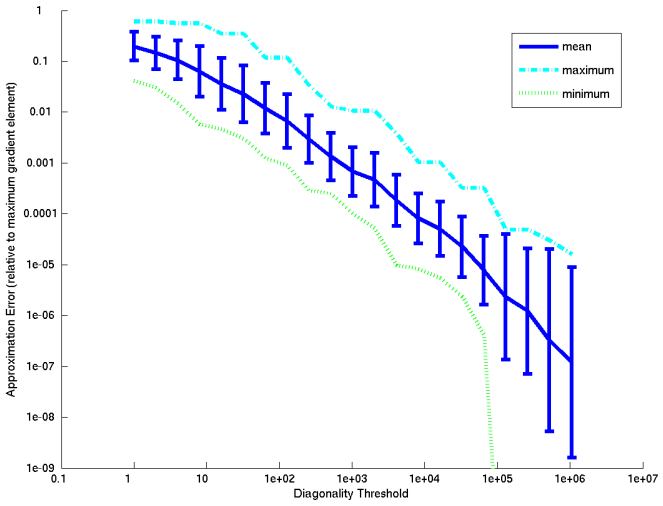


Figure 5.3: Approximation error vs. weight threshold for C sub-matrix.

When implemented in MATLAB®, the computational gains of this method come from reducing the size of the matrix inverse and reducing the size of the matrix multiplication. The matrix inverse does not dramatically dominate the computation for 512D bases, but takes 8 times longer than the matrix multiplication on 1024D bases. For small bases, the matrix multiplication is more expensive

than the matrix inverse. To test the accuracy of this approximation I used a 512D basis trained with KL-normalized coefficients and EGD basis updates. The accuracy of the approximation is roughly related to how close the C block of the matrix is to being a diagonal matrix with large elements. Figure 5.3.1 shows how the error relates to the weight threshold used for fifty high-loss examples from the MNIST database. The approximation is exact if the weights that are included in the C block are all zero, and this can provide an important computational savings over using the matrix inversion lemma.

5.4 Experiments

The performance of KL-sparse coding has been verified on several benchmark tasks including the MNIST handwritten digit recognition data-set, handwritten lowercase English characters classification, movie review sentiment regression, and music genre classification. In each application, the coefficients $\hat{W}$ produced using KL-regularization proved more useful for prediction than those produced with L_1 regularization due to the stability and differentiability provided by KL.

Sparsity

KL-regularization retains the desirable pseudo-sparsity characteristics of L_1, namely that each example, x, produces only a few large elements in $\hat{w}$. Figure 5.4 compares the mean sorted and normalized coefficient distribution over the 10,000 digit MNIST test set for KL-divergence and several L_p^p regularization functions, and shows that although the KL regularization function is not sparse in the traditional sense of setting many elements of $\hat{w}$ to zero, it is sparse in the sense that $\hat{w}$ contains only a few large elements in each example, lending support to the idea that this sense of sparsity is more important for classification.

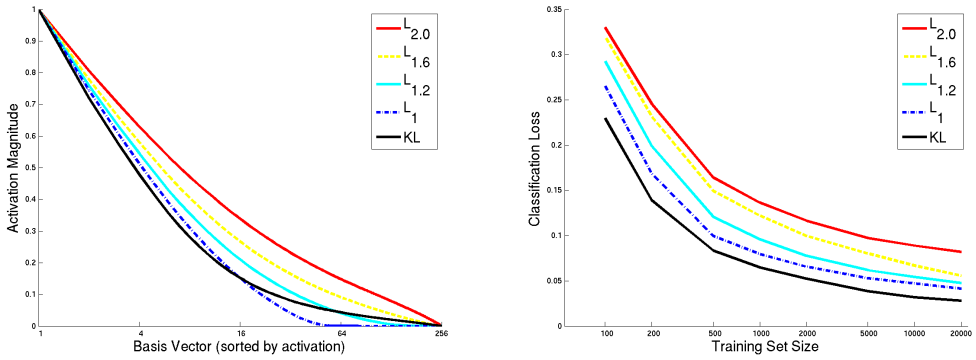


Figure 5.4: Left: Mean coefficient distribution over the 10,000 digit MNIST test set for various regularization functions. Each example $\hat{w}$ was sorted by magnitude and normalized by $\|\hat{w}\|_\infty$ before computing the mean over all examples. Right: test set classification performance. Regularization functions that produced few large values in each examples (such as KL and L1) performed best. However, forcing small coefficients to be exactly 0 was not necessary for good performance. Note the log scale on the horizontal axis.

Stability

The KL-divergence regularization function produces MAP estimates $\hat{w}$ that change smoothly with x and B. This stability is useful for prediction. Figure 5.5 shows the most-discriminative 2-D subspace of the coefficient space (as calculated by Multiple Discriminant Analysis [91]) for the input space, the L_1 and KL coefficient space, and the KL coefficient space after it has been specialized by back-propagation. The L_1 coefficients tame the disorder of the input space so that clusters for each class are apparent, although noisy and overlapping. The switch to KL regularization makes these clusters more distinct, and applying back-propagation further separates the clusters.

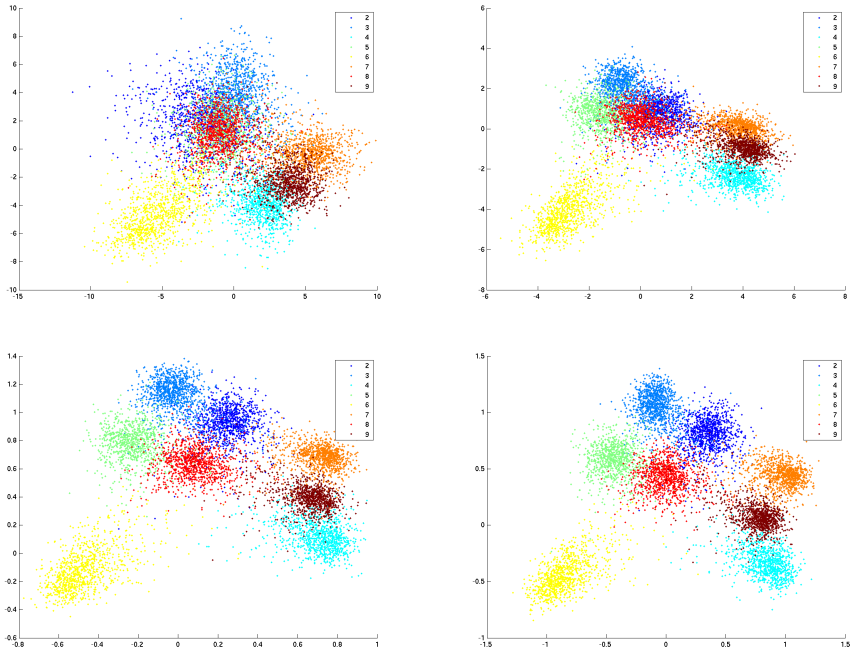


Figure 5.5: Shown is the distribution of the eight highest error digit classes in the most discriminative 2-D subspace (as calculated by Multiple Discriminant Analysis). The PCA-whitened input space (top left) contains a lot of overlap between the classes. L_1 regularization (top right) discovers structure in the unlabeled data, but still produces more overlap between classes than KL sparse approximation (bottom left) does with the same basis trained with L_1 sparse coding. Refining the basis with back-propagation on labeled examples further improves KL (bottom right). Figure best seen in color.

Improved Prediction Performance

On all four applications, the stability provided by KL-regularization improved performance over L_1, and back-propagation further improved performance when the training set had residual error after an output classifier was trained.

The first application used the benchmark MNIST handwritten digits data set [92] as shown in Figure 5.6. 10,000 of the 60,000 training examples were reserved for validation, and classification

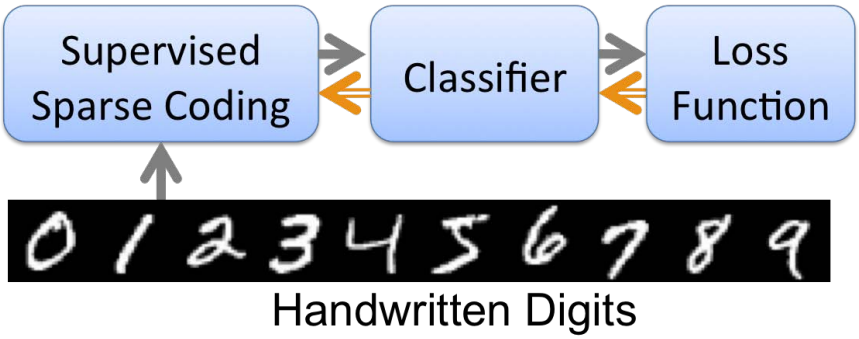


Figure 5.6: Handwritten Digits Classification

performance was evaluated on the separate 10,000 example test set. Each example was first reduced to 180D from 768D by PCA. Sparse coding was performed using a linear transfer function and squared loss[7]. The regularization constant, λ, and the prior mean for KL, p, were optimized on the validation set.

Maxent classifiers[8] [93] were learned on random subsets of the training set. Switching from L_1-regularized to KL-regularized sparse approximation improved performance for all subset sizes, with the improvement becoming relatively larger as the number of training examples increased (Table 5.1). When trained on all 50,000 training examples, the test set classification error of KL coefficients, 2.21%, was 37% lower than the 3.53% error rate obtained on the L_1-regularized coefficients. As shown in Table 5.2, this increase in performance was consistent across a diverse set of classification algorithms. After running back-propagation with the KL-prior, the test set error was reduced to 1.30%, which improves on the best results reported[9] for other shallow-architecture permutation-invariant classifiers operating on the same data set without prior knowledge about the problem[10], (see Table 5.3).

Transfer to Handwritten Character Classification

In [21], a basis learned by L_1-regularized sparse coding on handwritten digits was shown to improve classification performance when used for the related problem of handwritten character recognition with small training data sets (< 5000 examples). The handwritten English characters data set[12] they used consists of 16x8 pixel images of lowercase letters (Figure 5.7). In keeping with their work, we padded and scaled the images to match the 28x28 pixel size of the MNIST data, projected onto the

[7] This methodology was chosen to match [21]

[8] Also known as multi-class logistic regression

[9] An extensive comparison of classification algorithms for this data set can be found on the MNIST website, http://yann.lecun.com/exdb/mnist/

[10] Better results have been reported when more prior knowledge about the digit recognition problem is provided to the classifier, either through specialized preprocessing or by giving the classifier a model of how digits are likely to be distorted by expanding the data set with random affine and elastic distortions of the training examples or training with vicinal risk minimization. Convolutional Neural Networks produce the best results on this problem, but they are not invariant to permutations in the input since they contain a strong prior about how pixels are connected.

[12] Available at http://ai.stanford.edu/b̃taskar/ocr/

Training Set Size	1000	2000	10000	20000	50000
L1 (Test Set)	7.72%	6.63%	4.74%	4.16%	3.53%
KL (Test set)	5.87%	5.06%	3.00%	2.51%	2.21%
KL After Backprop (Test Set)	**5.66**	**4.46%**	**2.31%**	**1.78%**	**1.30%**
Improvement from Backprop	3.6%	11.9%	23.0%	29.1%	43.0%
KL (Training Set)	0.00%	0.05%	1.01%	1.50%	1.65%

Table 5.1: The ability to optimize the generative model with back-propagation leads to significant performance increases when the training set is not separable by the model learned on the unlabeled data. Shown is the misclassification rate on the MNIST digit classification task. Larger training sets with higher residual error benefit more from back-propagation.

Classifier	PCA	L_1	KL	KL+backprop
Maxent	7.49	3.53	2.21	**1.30**
2-layer NN	2.23	2.13	1.40	**1.36**
SVM (Linear)	5.55	3.95	2.16	**1.34**
SVM (RBF)	1.54	1.94	**1.28**	1.31

Table 5.2: The stability afforded by the KL-prior improves the performance of all classifier types over the L_1 prior. In addition back-propagation allows linear classifiers to do as well as more complicated non-linear classifiers.

Algorithm	L1	KL	KL+backprop	SVM	2-layer NN [94]	3-layer NN[11]
Test Set Error	3.53%	2.21%	**1.30%**	1.4%	1.6%	1.53%

Table 5.3: Test set error of various classifiers on the MNIST handwritten digits database.

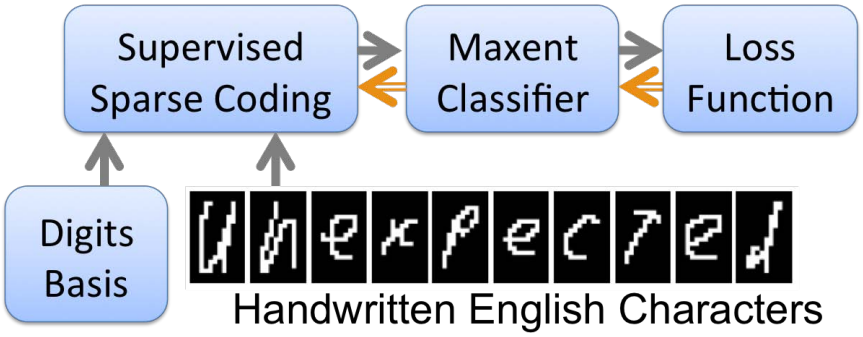


Figure 5.7: Transfer learning experiment using a basis learned on handwritten digits to classify English characters.

same PCA basis that was used for the MNIST digits, and learned a basis from the MNIST digits by L_1-regularized sparse coding. This basis was then used for sparse approximation of the English characters, along with a linear transfer function and squared loss.

In this application as well, Table 5.4 shows that simply switching to a KL prior from L_1 for sparse approximation significantly improves the performance of a maxent classifier. Furthermore,

the KL prior allows online improvement of the sparse coding basis as more labeled data for the character-recognition task becomes available. This improvement increases with the size of the training set, as more information becomes available about the target character recognition task.

Training Set Size	Raw	PCA	L1	KL	KL+backprop
100	44.3	46.9	44.0	49.4	**50.7**
500	60.4	61.2	63.7	69.2	**69.9**
1000	66.3	66.7	69.5	75.0	**76.4**
5000	75.1	76.0	78.9	82.5	**84.2**
20000	79.3	79.7	83.3	86.0	**89.1**

Table 5.4: Classification Accuracy on 26-way English Character classification task.

Comparison to sLDA: Movie Review Sentiment Regression

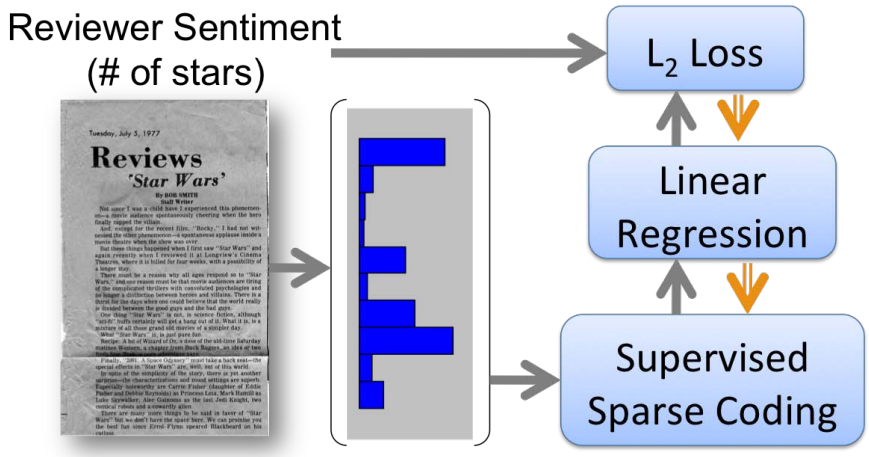


Figure 5.8: Movie review sentiment classification experiment. A histogram of 5000 words was computed from each review and used for sparse coding. The coded representation was then used with linear regression to predict the numerical rating the reviewer gave the movie.

KL-regularized sparse coding bears some similarities to the supervised LDA (sLDA) model introduced in [95], and we provide results for the movie review sentiment classification task [96] used in that work (Figure 5.8). To match [95] we use vectors of normalized counts for the 5000 words with the highest tf-idf score among the 5006 movie reviews in the data set, use 5-fold cross validation, compute predictions with linear regression on $\hat{w}$, and report our performance in terms of predictive R^2 (the fraction of variability in the out-of-fold response values which is captured by the out-of-fold predictions $\hat{y}$: $pR^2 := 1 - (\sum(y - \hat{y})^2)/(\sum(y - \bar{y})^2)$). Since the input is a probability distribution, we use a normalized exponential transfer function, $f(B, w) = \frac{e^{Bw}}{\|e^{Bw}\|_1}$, to compute the reconstruction of the input. For sparse coding we use KL-divergence for both the loss and the

regularization functions, as minimizing the KL-divergence between the empirical probability distribution of the document given by each input vector x and $f(B, w)$ is equivalent to maximizing the "constrained Poisson distribution" used to model documents in [97], where N is the number of words in the document (5.30).

$$P(X|Nr) = \prod_i \frac{e^{-Nr_i}(Nr_i)^{X_i}}{X_i!} \tag{5.30}$$

$$-log(P(X|Nr) = \sum_i Nr_i - X_i \log Nr_i + \log(X_i!) = N \sum_i r_i - \bar{x}_i \log r_i + C$$

$$\mathrm{KL}(\bar{x}||r) = \sum_i r_i - \bar{x}_i \log r_i + \bar{x}_i \log \bar{x}_i - x_i = \sum_i r_i - \bar{x}_i \log r_i + C \tag{5.31}$$

Table 5.5 shows that the sparse coding generative model we use is competitive with and perhaps slightly better than LDA. After back-propagation, its performance is superior to the supervised version of LDA, sLDA.

predictive R^2	Algorithm
0.263	LDA [95]
0.264	64D unsupervised KL sparse coding
0.281	256D unsupervised KL sparse coding
0.500	sLDA [95]
0.534	256D KL-regularized coding with backprop

Table 5.5: Results on the movie review sentiment classification task.

Music Genre Classification

The Music genre classification task was also used for L_1 sparse coding in [21], and consists of 15, 60-second musical clips from each of 17 different genres. Following their practice, each song was divided into 50 ms snippets, and the magnitude of the spectrogram for each snippet were used as input examples. For reconstruction, squared loss was used with a linear transfer function, and a maxent classifier was used for classification. The first 10 genres were used as unlabeled data to learn B via sparse coding, 10 clips from each of the other 7 genres were used as training data and 5 clips were used as testing data. As shown in Figure 5.6, KL-regularization improved classification performance on the same basis over L_1-regularization.

Training Set Size	PCA	L1	KL
200	32%	31%	33%
2000	43%	43%	45%
10000	48%	49%	50%

Table 5.6: Classification Accuracy on a 7-way music genre classification task is increased by using the KL prior for sparse approximation.

5.5 Further details on the KL prior

5.5.1 Sparse KL prior

The important stability benefits of the KL prior come from its slowly but monotonically increasing penalization of large weight values, not it's differentiability at 0. Consequently, the KL prior can be made sparse (in the sense that many components are zero) without compromising its usefulness for backpropagation by adding a small positive vector ξ to w in the KL term and adding an additional L1 penalization on w. This corresponds to simply shrinking each element of the weight vector by a constant and thresholding.

$$\Phi(w) = KL(w + \xi \| p) + \|w\|_1 \tag{5.32}$$

5.5.2 Matching KL to L1

The KL parameters λ, p can be set to match L1 regularization at specific weight magnitudes. Setting the two regularization penalties equal to each other: $\lambda|w_i| = \gamma w_i \log(\frac{w_i}{p}) - w_i$ produces $\gamma = \frac{\lambda}{\log(\frac{w}{p})-1}$. As p becomes smaller, KL provides better approximations to L_1. Experimentally, a value of $p = 0.005$ was found to be the smallest size that could be optimized efficiently with the EGD approach described in Section 5.3.

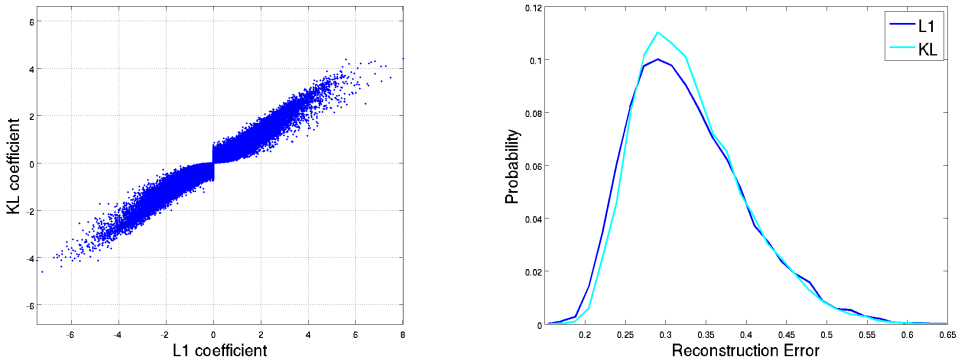


Figure 5.9: The KL regularization parameters were chosen to match the value of the L1 regularization penalty on the average non-zero weight size. A scatter-plot of the KL and L1 coefficients (Left) shows that there is much agreement, although the KL coefficients are less sparse at 0. The distribution of training set reconstruction errors is also similar (Right).

5.6 Practical Advice on Sparse Coding

This section is an attempt to take a step back from the theory discussed previously and present instead some intuition about what sparse coding can accomplish and how to use it. As will be

shown in Section 7.4 and previous work [98], sparse coding can be very effective for denoising signals that have been corrupted with noise that is i.i.d on each input dimension. The intuition here is that it is highly unlikely for an i.i.d noise vector to be correlated with any one vector that spans many dimensions of the input signal. The L_1 prior has a soft-thresholding effect on the activations of the basis vectors. Unless a basis vector is strongly correlated with the reconstruction residual, it will not participate in a minimum-energy solution. An L_2 prior on the other hand has a scaling effect on the basis vectors as discussed in Section 5.2.2, reducing the noise along with the signal without any hope of eliminating it completely. When sparse coding is used as a preprocessing stage for classification/regression, denoising by the sparse coding layer can translate into better generalization performance when inputs that only differ because of noise are mapped to the same internal representation before the prediction step. Effective denoising of this type can reduce the complexity required in the prediction step, and lead to better generalization performance from small amounts of labeled data.

Conversely sparse coding is not effective in situations with a large amount of correlated "noise" that exhibits regular structure in the input data. For instance consider the problem of watermark detection in real-world images and audio. Because the watermark is a small part of the overall signal, and the rest of the signal has lots of consistent structure, sparse coding would be a poor choice. The general rule is that sparse coding can be very powerful in situations where the desired information in the signal has strong regular structure, and the noise is on each input dimension is expected to be mutually uncorrelated.

5.6.1 Implementation Advice

Sparse coding can be difficult to use successfully because it involves setting several hyper-parameters, overcoming non-convex optimization difficulties, and it can be extremely computationally intensive. The main hyperparameters of sparse coding are the regularization constant(s) and the number of basis vectors. Sparse coding is generally not very sensitive to the number of basis vectors, as long as there is a sufficient number, and an effective strategy can be to double the size of the basis until the validation performance stops improving. The regularization parameter is harder to set, as it depends on the magnitude of "noise" in the input relative to the magnitude of the desired signal. It can be quickly adjusted to a reasonable range, however, by the appropriate visualization tools.

Good visualization tools are essential to the effective use of sparse coding. Because sparse coding operates by reconstructing the input, good visualizations for the original imput signal, the reconstructed version, and the basis vectors can be extremely helpful for understanding how the hyper-parameters can be changed to improve performance. For instance, if the reconstructed version of several examples is very small, it can indicate that the regularization constant is too large. Visualizing the basis vectors as shown in 5.1 can also be very helpful for diagnosing potential problems. If the basis vectors look exactly like particular input examples, it may indicate that the regularization constant is too large. On the other hand, if the basis vectors have a "noisy" appearance it may indicate that the regularization constant is too small. If there are more basis vectors than necessary, the set of basis vectors will often include basis vectors that are near-identical copies of each other. Good visualization tools are key for discovering the correct order of magnitude for each hyper parameter. Finer hyper-parameter optimization, though, currently requires extensive use of cross-validation. Fortunately, evaluating different hyper-parameter settings is a trivally parallel

problem, and a small investment in software infrastructure to support automatic parallel evaluation of different hyper-parameter settings can pay big dividends.

The inherent non-convexity of (fixed rank) sparse coding can often cause problems where a small number of basis vectors are used extensively, and the rest are not used at all. A common strategy for this situation is to replace under-used basis vectors with copies of input examples, or the reconstruction error from a particularly poorly reconstructed example. Another useful trick is to place a separate regularization constant on each basis vector, and increase the regularization penalty for frequently-used basis vectors (making the under-used vectors "cheaper" to use).

One last factor to keep in mind is the importance of starting small, with small bases and example sets, due to the high computational cost of sparse coding with large bases. Once good hyper-parameter settings are found it is easy to scale up to full-size problems.

5.7 Note on Multi-layer Sparse Coding

As discussed in Section 2.2, deep neural networks can produce powerful learning machines, possibly because later layers represent successively more abstract concepts. An interesting question then is if the same holds true for sparse coding techniques. Can a hierarchy of sparse coding modules produce a more powerful input representation for classification than a single layer?

Section 5.6 described how a single layer of sparse coding can provide useful denoting capabilities. However, a second linear layer of sparse coding would have little chance of performing additional denoising, as ultimately it would still be linearly reconstructing the input along independently-regularized dimensions. In [80], Karklin and Lewicki address this problem by instead sparse coding the covariance between dictionary elements used to represent the input. When applied to natural image patches, this mimics the properties of complex cells in the visual cortex [99]. Yang et al. [100], extended the methodology presented in this chapter to a deeper system where sparse coding is used between a local image descriptor function and a non-linear max-pooling transfer function that adds spatial invariance. The output of the max-pooling function is then classified by a linear SVM. A wider variety of image descriptor functions, pooling functions and output classifiers were considered by Boureau et al. [101] with a similar procedure. Their work found that sparse coding systematically outperforms hard and soft vector quantization as a coding method, and max-pooling transfer functions can be much more effective than average-pooling functions, particularly when paired with less complex output classifiers such as linear SVMs.

Unlike autoassociators and RBMs, a sparse coding module cannot be converted into a layer of a traditional supervised neural network, and hence they are hard to use purely as a pre-training tool. Stacking multiple layers is challenging computationally and theoretically, and while recent experiments by Kai Yu and others [102] have shown that adding a second layer of sparse coding can make classifier training more computationally efficient, it did not lead to an improvement in classification performance. The consensus of current research seems to be that discriminatively trained sparse coding modules are most useful as a single layer in a deeper system of heterogeneous learning modules.

Chapter 6

Learning from multiple related tasks

Chapter 5 presented results showing that pre-training a sparse coding module on an input reconstruction task can create a powerful semi-supervised learning algorithm. In that case input reconstruction served as an "auxiliary" task that was *related* to the target prediction task in that the same module parameters that were learned for input reconstruction, also proved useful for the prediction task. Input reconstruction is not the only potentially useful auxiliary task, however, and this chapter considers the case where many auxiliary tasks are available. Specifically, it will examine the problem of learning linear classifiers with limited training data and a large number of input features. Our goal is to construct an efficient online learning algorithm that can leverage related auxiliary tasks to guide *feature selection*, the process of selecting a small number of "relevant" features useful for solving the target task from the full input feature set. Ideally the regret of this learning algorithm should grow very slowly with the total number of input features.

The Exponentiated Gradient Descent (EGD) algorithm discussed in Section 2.1.4 has the desirable property that regret of the algorithm grows at most logarithmically with the number of irrelevant features. This is also true for mirror descent with the $\Phi(w) = \|w\|_p^2$ regularization function, where $w \in \Re^d$, and $p \approx 2\log(d)$ [28]. Both of these algorithms can perform automatic feature selection while only paying a slowly-growing penalty, however they only operate on a single task.

In many applications, there are several auxiliary tasks that could be equally effective, and it is desirable to be able to learn from all of them. Multiple auxiliary tasks have been used in recent research into Natural Language Processing [14], and image recognition [42] problems. In mobile robot navigation, the Crusher robot records the angle of the suspension arm of each wheel, which can then be used to calculate the height of the true supporting surface of the ground. The acceleration of the vehicle frame can be used to detect impacts with rigid obstacles. Teleoperated runs also provide additional information, and an auxiliary task could be to predict the speed at which an experienced operator will choose to drive the vehicle through a particular voxel.

This chapter presents experiments and novel theory for learning from supervised auxiliary tasks. In particular it will introduce an efficient online learning algorithm which learns from multiple related supervised tasks by defining a block-norm regularization function over the parameters for all the tasks which encourages feature sharing. The multi-task learning problem also turns out to be fundamentally related to the sparse coding problem discussed previously, and theoretical results developed in this chapter will find an additional application in the next Chapter as a convex opti-

mization method for sparse coding.

6.1 Multi-task Learning Through Group Regularization

Several recent papers have employed algorithms which benefit from learning multiple tasks simultaneously by defining a shared regularization function over multiple tasks and forcing the tasks to share a common subset of the input features [42, 103]. To formalize this, consider the problem of learning a linear classifier for each of k tasks. For each task j we are given a set of labeled input examples $\{(x_{j,1}, y_{j,1}), ..., (x_{j,n}, y_{j,n})\}$, and we seek to learn a weight vector W_j which minimizes:

$$\lambda\Phi(W) + \sum_{j,t} L_j(f(W_j, x_{j,t}), y_{j,t}), \tag{6.1}$$

where the columns of W are the weight vectors for the tasks, $\Phi(W)$ is the shared regularization function between the tasks, and L_j is the loss function for task j. The purpose of $\Phi(W)$ is to provide a "group discount" that encourages multiple tasks to share common input features.

A contribution of this thesis is showing that the mirror descent framework discussed in Section 2.1.2 can be used to create an efficient online multi-task learning algorithm using the $\Phi(W) = \frac{1}{2}\|W\|^2_{p,q}$ *compositional norm* to implement the shared regularization $\Phi(W)$, where $p \geq 2$ and $1 < q < 2$. A limiting case of this norm, the $L_{2,1}$ block norm has been advocated recently as a regularization function for *multi-task learning* [104, 105]. To review, the mirror descent update rule provides an efficient way of exactly optimizing a linear approximation to a series of convex loss functions with the update rule (2.7):

$$\hat{w}_t = \nabla\Phi^*\left(-\frac{1}{\lambda}\sum_t \nabla L_t(w)\right). \tag{6.2}$$

I will show that the *Fenchel conjugate* of $\Phi(W)$ is $\Phi^*(W) = \frac{1}{2}\|W\|^2_{p^*,q^*}$, where $\frac{1}{p}+\frac{1}{p^*} = \frac{1}{q}+\frac{1}{q^*} = 1$, by proving a general norm duality theorem for compositional norms. I will also describe how the potential-based gradient descent framework can be generalized to non-differentiable regularization functions, where it leads to a boosting approach.

6.2 Compositional Norms

A *compositional norm* is a norm composed of norms over disjoint groups of variables[1]. Given a column vector $x \in \Re^n$, a collection of disjoint sets of indices $I = \{s_1, \ldots, s_d\}$ such that each integer $i \in \{1, \ldots, n\}$ is in exactly one set s_j, a set of norms $S = \{\|\cdot\|_{s_1}, \ldots, \|\cdot\|_{s_d}\}$, and a top-level norm $\|\cdot\|$, the compositional norm $\|x\|_{C(I,S,\|\cdot\|)}$ is defined as:

$$\begin{aligned} &\|x\|_{C(I,S,\|\cdot\|)} = \|\alpha\| \\ &\quad \text{where } \alpha_j = \|x_{i\in s_j}\|_{s_j}, \qquad \forall j \in \{1, \ldots, d\} \end{aligned} \tag{6.3}$$

[1]The component norms of a compositional norm can also be compositional norms, allowing hierarchical arrangements of three or more norms.

A useful and notationally convenient example of a compositional norm is a block norm[2]. Define a block norm of the matrix W, $\|W\|_{p,q}$ to be the L_q norm of the L_p norms of every row W^i:

$$L_{p,q}(W) = \|W\|_{p,q} = \left(\sum_i \left(\sum_j |W_j^i|^p \right)^{\frac{q}{p}} \right)^{\frac{1}{q}} \tag{6.4}$$

A block norm $\|X\|_{p,q} = \left(\sum_i \|Z^i\|_p^q\right)^{1/q}$, can be written as a compositional norm where x is formed by concatenating the rows of X into a column vector. Each set $s_i \in I$ is the indices of the variables in x corresponding to row i of X. Each norm $\|\cdot\|_{s_i} \in S$ is the L_p norm $\|\cdot\|_p$, and the top level norm is $\|\cdot\|_q$. As each column of W is the weight vector for a separate task, applying a norm across the rows of W with a block norm can provide a "group discount" that encourages the tasks to share a common subset of the features. In the extreme case of an L_∞ norm across the rows, the penalty for incorporating a feature is completely determined by its maximum weight on any task, and there is no additional penalty for other tasks to use that feature. As discussed in Section 2.1.2, the online mirror-descent framework requires that the regularization function $\Phi(W)$ be a Legendre function with a known Legendre dual $\Phi^*(W)$. A more general version of the Legendre dual is the Fenchel-Legendre conjugate (6.5), also known as the conjugate function [29], which generalizes Legendre duality to include non-differentiable functions:

$$f^*(z) = \sup_x \left\{ x^T z - f(x) \right\}. \tag{6.5}$$

Here $f^*(z)$ is the conjugate of the function $f(x)$, and the variable z is the dual variable of x. It is well known that the Fenchel conjugate of a squared norm $f(x) = \frac{1}{2}\|x\|^2$ is simply the square of its *dual norm*. Let $\|\cdot\|$ be a norm on R^n The dual norm, $\|\cdot\|_*$ is defined as:

$$\|z\|_* = \sup_{\|x\| \le 1} z^T x \tag{6.6}$$

For norms, the dual of the dual norm is the original norm $\|x\|_{**} = \|x\|$. The dual of the L_1-norm $\|x\|_1$ is the L_∞-norm $\|z\|_\infty$, and the dual of an L_p-norm $\|x\|_p$ is the L_q-norm $\|x\|_q$, where $1/p + 1/q = 1$ [29]. Hence by establishing the following lemma about the dual of a block norm, we also find the Fenchel conjugate of the squared block norm, and an online multi-task learning algorithm:

Lemma 1 *The dual of the $L_{p,q}$ block norm is a L_{p^*,q^*} block norm where $1/p+1/p^* = 1/q+1/q^* = 1$, $p, p^*, q, q^* \in [1, \infty]$.*

6.2.1 Dual of a Compositional Norm

Lemma 1 is a special case of a more general lemma about the dual of a compositional norm:

Lemma 2 *The dual of a compositional norm, $\|x\|^*_{C(I,S,\|\cdot\|)}$ is a compositional norm $\|z\|_{C(I,S^*,\|\cdot\|^*)}$, where the top-level norm and each norm in S has been replaced by its dual-norm:*

$$\begin{aligned} \|x\|^*_{C(I,S,\|\cdot\|)} &= \|z\|_{C(I,S^*,\|\cdot\|^*)} = \|\beta\|^* \\ &\text{where } \beta_j = \|z_{i \in s_j}\|^*_{s_j}, \qquad \forall j \in \{1, \ldots, d\} \end{aligned} \tag{6.7}$$

[2] It can be easily verified that (6.4) satisfies the definition of a norm.

The proof of this lemma follows readily from the definition of the dual norm (6.6) and the fact that each norm in S operates on a disjoint set of variables. The dot product in (6.6) can be rewritten as the sum of the dot products of each set of variables (6.8). Next, by the definition of the compositional norm (6.3), the constraint $\|x\|_{C(I,S,\|\cdot\|)} \leq 1$ can be replaced by a constraint on each set of variables, $\|x_{i\in s_j}\|_{s_j} \leq \alpha_j$, and a constraint on α, $\|\alpha\| \leq 1$ (6.9). By the positive scalability property of norms, the constraint $\|x_{i\in s_j}\|_{s_j} \leq \alpha_j$ can be rewritten as $\|\alpha_j x_{i\in s_j}\|_{s_j} \leq 1$ (6.10). However, (6.10) is the definition of the dual norm of $\alpha_j z_{i\in s_j}$, and by using positive scalability again the α_j factor can be moved outside the dual norm (6.11). Finally we substitute $\beta_j = \|z_{i\in s_j}\|^*_{s_j}$, recognize (6.12) as the dual of the top-level norm and recover (6.7).

$$
\begin{aligned}
\|z\|^*_{C(I,S,\|\cdot\|)} &= \sup_{\|x\|_{C(I,S,\|\cdot\|)}\leq 1} z^T x \\
&= \sup_{\|x\|_{C(I,S,\|\cdot\|)}\leq 1} \sum_j z^T_{i\in s_j} x_{i\in s_j} && (6.8)\\
&= \sup_{\|\alpha\|\leq 1} \sum_j \sup_{\|x_{i\in s_j}\|_{s_j}\leq \alpha_j} z^T_{i\in s_j} x_{i\in s_j} && (6.9)\\
&= \sup_{\|\alpha\|\leq 1} \sum_j \sup_{\|x_{i\in s_j}\|_{s_j}\leq 1} \alpha_j z^T_{i\in s_j} x_{i\in s_j} && (6.10)\\
&= \sup_{\|\alpha\|\leq 1} \sum_j \alpha_j \|z_{i\in s_j}\|^*_{s_j} && (6.11)\\
&= \sup_{\|\alpha\|\leq 1} \sum_j \alpha_j \beta_j && (6.12)\\
&= \|\beta\|^*, \qquad \beta_j = \|z_{i\in s_j}\|^*_{s_j}\ \forall j \in \{1,\ldots,d\}
\end{aligned}
$$

As an aside, this proof can be easily extended to compositional norms with more than two layers of norms, as long as each input variable has only one "path" to the top-level norm (i.e. the norms in each layer use disjoint subsets of the variables used by the norm above them). Since a compositional norm satisfies all of the properties of a standard norm, compositional norms can be used for some or all of the first-level norms in S. In this way, Lemma 2 can be used to find the dual of a compositional norm with an arbitrary number of layers.

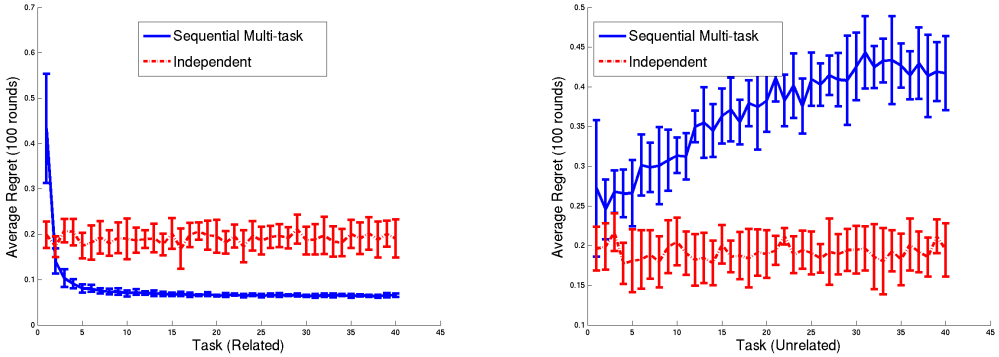
6.3 Online Multi-task learning with Compositional Norms

Armed with Lemma 1, we can now construct an online multi-task learning algorithm that uses mirror descent (2.7). The update rule comes from the partial derivative of the Fenchel conjugate $\Phi^*(Z) = \frac{1}{2}\|Z\|^2_{p^*,q^*}$ with respect to feature i and task j:

$$
\frac{\partial \Phi^*(Z)}{\partial Z^i_j} = \frac{\|Z^i\|^{q^*-p^*}_{p^*} \cdot |Z^i_j|^{p^*-1} \cdot \text{sign}(Z^i_j)}{\|Z\|^{q^*-2}_{p^*,q^*}} \tag{6.13}
$$

Substituting ${}^tZ_j = -\frac{1}{\lambda}\sum_{t' \le t} \nabla L_{j,t}(W_j)$ produces the following update rule for the weight ${}^tW_j^i$ on feature i in task j after t examples have been observed:

$$ {}^tW_j^i = \frac{\partial \Phi^*({}^tZ)}{\partial {}^tZ_j^i} = \frac{\|{}^tZ^i\|_{q_1}^{q_2-q_1} \cdot |{}^tZ_j^i|^{q_1-1} \cdot \text{sign}({}^tZ_j^i)}{\|{}^tZ\|_{q_1,q_2}^{q_2-2}} \tag{6.14} $$



A B

Figure 6.1: Left: Performance vs. number of related tasks. Right: Performance Vs. number of unrelated tasks

By setting p to 2 and q close to 1, the block norm regularization function $\Phi(W) = \|W\|_{p,q}^2$ can be used to promote feature sharing across tasks. In tests on synthetic data, the resulting soft rank constraint on the number of features used has proven useful for learning related tasks which all use a small subset of a large input space. Figure 6.1A, shows a situation where tasks that used the same four input variables (out of 1000 total input dimensions) benefited from using a matrix prior with $p = 2$ and $q = 1.2$. Here the tasks were presented sequentially, meaning that all of the examples for the first task were presented before the first example for the second task. The matrix prior helps later tasks focus on only the relevant subset of variables and leads to much lower regret for the later tasks than learning the tasks individually.

In this experiment each task was generated by randomly picking a ground truth weight vector U_j with unit L_2 norm. 100 input feature vectors X_j were then drawn for each task from a binary distribution $X_j^i \in \{-1, 1\}$, and Gaussian noise was added to create the example labels $Y_j = X_j^T U_j + \sigma$. Constant learning rates were optimized for both the sequential multi-task learning (block-norm regularization) approach and a baseline where the tasks were learned independently with an $\|W_j\|_{1.2}$ prior on the task weight vector. A larger step size proved to be optimal for the sequential multi-task algorithm because the update rule (6.14) is resistant to noise in variables that were not picked by previous tasks. This larger step size led to large regret on the first task, but lowered regret on later tasks.

When the tasks are completely unrelated and have no features in common there can be a linear regret penalty in the number of tasks. Figure 6.1B shows the result of modifying the previous experiment so that each task uses a different set of four features. In this case the average regret of later tasks increases because unrelated features used by previous tasks have a low regularization

penalty. This last result demonstrates the importance of carefully selecting the auxiliary tasks, and complicates any attempt to provide useful regret bounds for online multi-task learning.

6.4 Extension to Legendre Functions of Norms

The techniques used to prove Lemma 2 can also be used to find the Fenchel conjugate of any sum of Legendre functions of norms over disjoint sets of input variables. As described in Section 2.1.2, Legendre regularization functions can produce powerful online learning algorithms if their conjugate functions are known. In particular the KL-divergence regularization function $KL(w\|p) = \sum_i w_i \log \frac{w_i}{p_i} - w_i + p_i$ produces the Exponentiated Gradient Descent (EGD) algorithm which is robust to irrelevant input features. KL-divergence regularization can be applied to group feature selection in multi-task learning by regularizing by the KL-divergence of a norm over each row W^i of the weight matrix W, producing the regularization function:

$$\Phi(W) = \sum_i \|W^i\| \log \frac{\|W_i\|_2}{p_i}, \qquad \sum_i \|W_i\| = \sum_i p_i = 1 \tag{6.15}$$

Building on the notation used in Section 6.2, consider the Legendre function G that is a sum of applying a set of Legendre functions f_j to norms of disjoint sets of input variables:

$$\begin{aligned} G(x, I, S, \|\cdot\|) &= \sum_j f_j(\alpha_j) \\ &\text{where } \alpha_j = \|x_{i \in s_j}\|_{s_j}, \qquad \forall j \in \{1, \ldots, d\}. \end{aligned} \tag{6.16}$$

Applying mirror descent to this class of regularization functions first requires Lemma 3:

Lemma 3 *The dual of $G(x, I, S, \|\cdot\|) = \sum_j f_j(\alpha_j)$ is $G^*(z, I, S, \|\cdot\|^*) = \sum_j f_j^*(\beta_j)$, where each $f_j(\alpha_j)$ is a legendre function, $f_j^*(\beta_j)$ is its conjugate, and $\alpha_j = \|x_{i \in s_j}\|_{s_j}$, $\beta_j = \|z_{i \in s_j}\|_{s_j}^*$, $\forall j \in \{1, \ldots, d\}$.*

To prove Lemma 3, we start by noticing that the Fenchel conjugate, G^* is a sum of separate conjugate problems, one for each disjoint set of variables in S:

$$\begin{aligned} G^*(z) &= \sup_x z^T x - G(x) \\ &= \sup_x \sum_j z_{i \in s_j}^T x_{i \in s_j} - f(\alpha_j) \\ &= \sum_j \sup_{x_{i \in s_j}} z_{i \in s_j}^T x_{i \in s_j} - f(\alpha_j) \\ &\text{where } \alpha_j = \|x_{i \in s_j}\|_{s_j}, \qquad \forall j \in \{1, \ldots, d\}. \end{aligned} \tag{6.17}$$

Because the function $f_j(\alpha_j)$ in each of these conjugate sub-problems is Legendre, there will be a finite maximum value $\beta_j = \sup_x z_{i \in s_j}^T x_{i \in s_j} - f(\alpha_j)$ for each sub-problem at some value $\hat{\alpha}_j = \|\hat{x}_{i \in s_j}\|_{s_j}$, and $f_j(\alpha_j)$ is guaranteed to be an increasing function of α_j at $\hat{\alpha}_j$. Hence, by taking advantage of the positive scalability of norms and recognizing the definition of the norm dual and the Legendre dual, (6.17) can be rewritten as:

$$
\begin{aligned}
G^*(z) &= \sum_j \sup_{\alpha_j} \left(\sup_{\|x_{i\in s_j}\|_{s_j} \le \alpha_j} z_{i\in s_j}^T x_{i\in s_j} \right) - f(\alpha_j) \\
&= \sum_j \sup_{\alpha_j} \left(\sup_{\|x_{i\in s_j}\|_{s_j} \le 1} z_{i\in s_j}^T x_{i\in s_j} \right) \alpha_j - f(\alpha_j) \\
&= \sum_j \sup_{\alpha_j} \|z_{i\in s_j}\|_{s_j}^* \alpha_j - f(\alpha_j) \\
&= \sum_j f^*(\|z_{i\in s_j}\|_{s_j}^*)
\end{aligned} \tag{6.18}
$$

Thus proving Lemma 3 and showing that the dual of a sum of Legendre functions of disjoint norms, is a sum of Legendre duals of the dual norm.

6.4.1 Example: KL-divergence group regularization

As an example, consider the normalized KL-divergence of L_2 group regularization function given in (6.15). The fenchel conjugate of this regularization function is:

$$
\Phi^*(Z) = \log\left(\sum_i p_i e^{\|Z^i\|_2 - 1} \right), \qquad \sum_i p_i = 1 \tag{6.19}
$$

The partial derivative of this Fenchel conjugate with respect to feature i and task j, provides the update rule:

$$
W_j^i = \frac{\partial \Phi^*(Z)}{\partial Z_j^i} = \frac{Z_j^i}{\|Z^i\|_2} \frac{p_i e^{\|Z^i\|_2 - 1}}{\sum_i p_i e^{\|Z^i\|_2 - 1}} \tag{6.20}
$$

Where ${}^tZ_j = -\frac{1}{\lambda} \sum_{t' \le t} \nabla L_{j,t}(W_j)$ after t examples have been observed.

6.4.2 Regret bounds for regularization functions composed of norms

Unfortunately regularization functions composed of norms, such as (6.4) and (6.15), are not strictly convex, and hence the regret bounding machinery described in Section 2.1.3 is not applicable. That machinery bounds the step size between examples in terms of the Bregman Divergence $D_\Phi(X_t \| X_{t+1})$ associated with the regularization function Φ. A key property that makes this possible is that $D_\Phi(X_t \| X_{t+1}) > 0$ whenever $X_t \neq X_{t+1}$. However, when the regularization function is a function of norms of disjoint sets of variables, the values of the variables in each disjoint set can be changed without changing the value of the norm. Hence this key positive distance requirement does not hold, and a valid Bregman divergence does not exist for these functions. New machinery is needed to provide regret bounds for the online learning algorithms derived from these regularization functions.

6.5 Information Theoretic Limits of Auxiliary Task Selection

The results of Section 6.3 highlight the need to be tolerant to unrelated tasks in multi-task learning. Ideally we wish to start with a large pool of m auxiliary tasks, only some of which may be related to the target task, and use a subset of those tasks to learn with low regret with respect to any fixed expert $u \in R^d$ on the target task. The regret $R_t(u)$ in this case is the difference in total loss between the series of hypotheses selected by the learning algorithm, $\widehat{L}_t$, on the first t examples and the loss suffered by the fixed hypothesis (expert) u, $L_t(u)$:

$$R_t(u) = \widehat{L}_t - L_t(u)$$

Define S^* to be the subset of tasks that provide the greatest multi-task learning benefit, and $k = |S^*|$ to be the number of tasks in that subset. Then we can rewrite the regret as (6.21).

$$\begin{aligned} R_t(u) &= \widehat{L}_t(u) - \min_{|S|} \widehat{L}_{|S|,t} + \min_{|S|} \widehat{L}_{|S|,t} - \min_{S \in |S|} \widehat{L}_{S,t} + \min_{S \in |S|} \widehat{L}_{S,t} - L_t(u) \\ &= R(|S^*|) + R_{|S^*|}(S^*) + R_{S^*}(u) \end{aligned} \tag{6.21}$$

Where $R(|S^*|)$ is the regret incurred to find the number of tasks in the best subset, $R_{|S^*|}(S^*)$ is the regret incurred to find the best subset containing a particular number of tasks, and $R_{S^*}(u)$ is the regret incurred with respect to the fixed expert u, given a particular set of tasks to use for multi-task learning. It is encouraging to see that the total regret for the online multi-task learning problem breaks down into a sum of regrets of simpler learning problems. The subset selection problem could be solved (at an immense cost in computational resources) by running two layers of exponentially weighted average forecasters. In this case the regret of each layer is bounded by $\sqrt{(t/2)\log N}$, where N is the number of hypotheses considered by the layer (Theorem 2.2 of [28]). This produces the regret bound:

$$\begin{aligned} R_t(u) &\leq \sqrt{2t\ln(m)} + \sqrt{2t\ln\binom{m}{k}} + R_{S^*}(u) \\ &\leq \sqrt{2t\ln(m)} + \sqrt{2tk\ln\left(\frac{me}{k}\right)} + R_{S^*}(u). \end{aligned} \tag{6.22}$$

Hence we pay only as the square root of the size of the relevant set of tasks, for a potentially superlinear increase in the number of effective samples. It is not clear how to achieve this bound with a computationally efficient online algorithm, but the mathematical framework developed for multi-task online learning also allows the creation of a novel, deterministic training procedure for sparse coding that is convex up to an oracle step.

6.6 Connection to Sparse Coding

Mathematically, multi-task learning is similar to the input coding problem discussed in Chapter 5, where each column W_j of W approximated an input example X_j as a weighted combination of the

columns of B:

$$\begin{aligned} \min_{W,B} \quad & L(X, f(BW)) + \lambda\Phi(W) \\ s.t. \quad & \|B_i\|_2 = 1, \ \forall i. \end{aligned} \tag{6.23}$$

In the multi-task learning setting the goal is to match a vector of N outputs $Y_j \in \Re^N$ for N labeled examples provided for each task j. Instead of shared basis vectors there are shared input vectors for each example that are arranged as columns X_t of a matrix X. When the goal is to learn a linear classifier $f(X_t^T W_j) \approx Y_j^t$ for each task j, the mathematical form of multi-task learning with shared regularization looks very close to the sparse coding formulation:

$$\min_W L\left(Y, f(X^T W)\right) + \lambda\Phi(W), \tag{6.24}$$

with the target outputs Y replacing the input matrix X, and X^T replacing the basis matrix B. The next chapter will explore how the compositional norms developed for online multi-task learning can be used for a convex (up to an oracle) version of sparse coding.

Chapter 7

Convex Coding

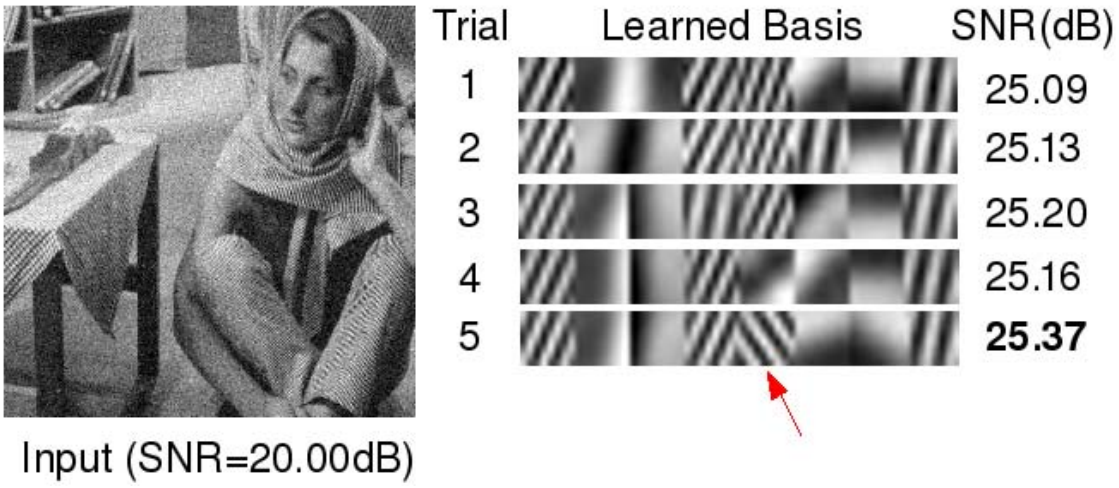


Figure 7.1: **Left:** input image. **Right:** five bases learned by alternating optimization starting from random initializations. The basis that performed best found an important pattern (indicated) that was not representable in the other bases, which were stuck in worse local minima.

Previous work in sparse coding[1] used non-convex methods to optimize the basis matrix B, and the coefficient matrix W. An important question is whether these non-convex optimization strategies return local minima that are inferior to the global minima. Figure 7.1 shows an example where this happened on the image denoising task examined in Section 7.4. One of the five random initializations lead to a basis that captured an important pattern in the image missed by the other four.

Clustering can be viewed as a restricted form of the coding matrix factorization problem [107]; a special case of coding where the basis vectors are the cluster centroids and the W matrix is the cluster membership of each example. Recently [108] showed that the clustering problem can be made convex by considering only a fixed set of possible cluster centroids. Exemplars are a natural choice

[1]Note, however, the interesting approach to convex sparse coding [106] that was developed in parallel with this work

for these candidate cluster centroids, but [109] shows that for some problems using only exemplars can be overly restrictive, and better results can be achieved by defining the problem in terms of a convex "master" problem, and a subproblem where new centroid candidates are generated.

This "convex clustering" approach can be extended to the coding setting. Starting from a convex but intractable version of sparse coding, I will derive a boosting-style approach that is convex except for a subproblem, give an efficient algorithm for solving the subproblem that will only improve on a fully convex exemplar-based approach, and demonstrate the resulting algorithm on an image denoising task.

This section contains three main contributions: first, the coding problem is split into a convex "master problem" and a subproblem where candidate basis vectors are generated [109]. Second, I provide a general class of boosting algorithms for matrix norms, and show how they can be derived with the Fenchel conjugate. Third, I provide experimental results that the alternating optimization strategy performs well in practice[2].

The following section presents a convex relaxation of the rank-constrained L_1 sparse coding problem (5.5). Recently [106] independently developed a similar convex regularization scheme for sparse coding, and they employ an interesting optimization strategy based on further convex relaxations that is complementary to the approach presented here. Section 7.2.2 derives a boosting-style algorithm to efficiently optimize this relaxation using the Fenchel conjugate. Section 7.3 analyzes the oracle subproblem, and presents a heuristic solution that is shown to be effective experimentally in Section 7.4. Note that although the resulting algorithm is deterministic, because an optimal solution to the oracle is not available, there is no guarantee that the algorithm described in Section 7.3 will return the optimal solution. The "Convex Coding" title of this chapter was chosen to emphasize the link with the "convex clustering" algorithm of [108].

7.1 Convex Relaxation of Sparse Coding

The L_1-regularized formulation given in (5.5), and similar variants, is commonly solved by alternating between optimization over W and optimization over B, as both problems are convex when the other matrix is constant. However, a common objection to this approach is that the joint optimization problem is non-convex.

As noted by [19] for the related problem of learning neural networks, the non-convexity can be removed if B is a fixed, infinite basis matrix containing all unit-length vectors as columns, and the optimization is only with respect to W. They go on to show that if L_1 regularization is placed on W, it will have optimal solutions with only a finite number of non-zero weights, even if the number of basis vectors is infinite. Hence, the matrix B in (5.5) can be interpreted as the small set of basis vectors that have non-zero weight in W, and the fixed number of columns of B can be written as a *compositional norm* constraint on W, using the compositional norms described in Section 6.2.

Since in our setting W is defined so that each row corresponds to a basis vector and each column corresponds to an example, a block norm can encourage all examples to use a subset of the basis vectors. For instance, the fixed size of B in (5.5) is equivalent to a hard constraint on W in terms of

[2]Sparse coding strategies using alternating optimization are often believed to be unreliable because of their hyperparameters and inherent non-convexity. In these experiments, non-convexity has only a minor impact on the quality of the solution.

the non-convex $L_{2,0}$ block semi-norm. $L_{2,0}$ is the L_0 semi-norm[3] of the L_2 norm of each row of W, which counts how many basis vectors have non-zero entries in W.

Relaxing this non-convex $L_{2,0}$ constraint by substituting regularization with the convex (but still sparse) $L_{2,1}$ constraint produces the modified sparse coding objective function:

$$\arg\min_W \frac{1}{2}\|BW - X\|_F^2 + \lambda\left(\frac{1}{2}\|W\|_{2,1}^2 + \frac{\gamma}{2}\|W\|_1^2\right), \tag{7.1}$$

The fact that the norms are squared in (7.1) will be mathematically convenient later, and is equivalent to scaling the regularization constant[4]. The $L_{2,1}$ block norm has been advocated recently as a regularization function for *multi-task learning* [104, 105].

7.2 A Boosting Approach to Coding

With a finite basis matrix B, (7.1) can be solved directly with various convex optimization techniques, including, e.g., subgradient[5] descent [27]. The regularization term effectively "selects" a small active set from the full basis (by encouraging multiple examples to share the same bases) to form the coded representation of the input. However, by exploiting properties of the $L_{2,1}^2 + \gamma L_1^2$ regularization function it is also possible to implicitly handle an infinitely large B. In the following I will extend the results of Chapter 6 to solve (7.1) with an efficient boosting algorithm (Algorithm 3) that adds one new basis vector to the active basis matrix in each step.

This approach is motivated by the view of boosting as functional gradient descent in the space of weak learners [17, 18]. In sparse coding, each weak learner is a vector with unit length. Each boosting step, attempts to maximize the correlation between the negative loss gradient and a "small" change in W, as measured by a regularization function $\Phi(W)$. For infinite sets of potential basis vectors, the maximization at each step of this boosting approach is a non-convex sub-problem which must be solved (or approximated) by an oracle (Algorithm 4).

7.2.1 Fenchel Conjugate

We start by recalling the definition of the Fenchel-Legendre conjugate, which generalizes Legendre duality to include non-differentiable functions:

$$f^*(z) = \sup_x \left\{x^T z - f(x)\right\}. \tag{7.2}$$

$f^*(z)$ is the conjugate of the function $f(x)$, and the variable z is the dual variable of x. When the supremum in (6.5) is achieved, every maximal value $\hat{x}$ of (6.5) is a subgradient with respect to z of the conjugate function $f^*(z)$:

$$\frac{\partial f^*(z)}{\partial z} = \hat{x} = \arg\max_x \left\{x^T z - f(x)\right\}. \tag{7.3}$$

[3] For $0 \le p < 1$, the L_p norm of a vector x is redefined as: $\|x\|_p = \sum_i |x|_i^p$

[4] At the minimum $\hat{W}$ of (7.1), $\lambda\|\hat{W}\|_{2,1}^2 = \lambda'\|W\|_{2,1}$, where $\lambda' = \lambda\|\hat{W}\|_{2,1}$ is the original regularization constant scaled by the $L_{2,1}$-norm of $\hat{W}$.

[5] A vector $\phi \in R^n$ is a subgradient, $\phi \in \partial_x f(x)$, of a function $f : R^n \to (-\infty, \infty]$ at $x \in R^n$ if $\phi' y \le f(x+y) - f(x)$, $\forall y \in R^n$. Consider the function $f = \max(x_1, x_2)$. If $x_1 > x_2$, then there is a unique gradient $\frac{\partial f}{\partial x} = [1\ 0]$. However, if $x_1 = x_2$, then [1 0] and [0 1] (and any convex combination of the two) are subgradients. $\partial_x f(x)$ is the set of all subgradients.

7.2.2 Boosting With Fenchel Conjugates

Each step of a gradient boosting style algorithm [17, 18] seeks a descent direction which provides the greatest reduction of loss for a small increase in a regularization function $\Phi(w)$. If w is a vector of weights over all of the possible weak learners, we wish to find the step $\Delta\hat{w}$ that is both maximally correlated with the negative loss gradient $\nabla L(w)$, and smaller than ϵ as measured by the regularization function $\Phi(w)$:

$$\Delta\hat{w} \quad = \quad \arg\max_{\Phi(\Delta w)\leq\epsilon} \; -\nabla L(w)^T \Delta w. \tag{7.4}$$

This section will show that if the regularization function $\Phi(w)$ is convex, defined on $\Re^d$, and the constraint is strictly feasible[6], every subgradient of its Fenchel conjugate evaluated on the loss gradient, $\Delta\hat{w} \in \partial\Phi^*(-\frac{1}{\lambda}\nabla L(w))$, is an optimal boosting step according to (7.4). This provides a useful method for constructing boosting algorithms for a wide class of regularization functions, and we will apply it to the convex sparse coding objective function (7.1).

First we upper bound (7.4) by the minimum of the unconstrained Lagrange dual function, assuming the step size constraint is feasible (i.e. there exists a Δw such that $\Phi(\Delta w) \leq \epsilon$):

$$\min_{\lambda\geq 0} \max_{\Delta w} \; -\nabla L(w)^T \Delta w - \lambda \left(\Phi(\Delta w) - \epsilon\right). \tag{7.5}$$

If Φ meets the conditions above, then the pair $(\Delta\hat{w}, \hat{\lambda})$ that optimizes the upper bound (7.5), will also be optimal for the primal (7.4), iff the KKT conditions (generalized to subdifferential functions) are satisfied [110]. In this case the conditions state that the negative loss gradient must be parallel to a subgradient of the regularization function (7.6), and all of the active constraints on $\Delta\hat{w}$ must be tight (7.7):

$$-\nabla L(w) \in \hat{\lambda}\partial\Phi(\Delta\hat{w}) \tag{7.6}$$

$$\hat{\lambda}\left(\Phi(\Delta\hat{w}) - \epsilon\right) = 0. \tag{7.7}$$

Since $\nabla L(w)^T \Delta w$ is a linear function of Δw and there is only one constraint, it must be active (i.e. $\lambda > 0$) whenever $\nabla L(w) \neq 0$. Since boosting would stop if $\nabla L(w) = 0$, dividing (7.5) by λ and adding ϵ does not change the optimal values of $(\Delta\hat{w}, \hat{\lambda})$, and produces the definition of the Fenchel conjugate of the regularization function, with the dual variable $z = -\frac{1}{\lambda}\nabla L(w)$:

$$\max_{\Delta w} \left\{ -\frac{1}{\lambda}\nabla L(w)^T \Delta w - \Phi(\Delta w) \right\} = \Phi^*(z). \tag{7.8}$$

Further, (7.3) means that all subgradients $\Delta\hat{w} \in \partial_z\Phi^*(z)$ are boosting steps which will optimize (7.4). Hence for convex functions we find a boosting update rule:

$$\Delta\hat{w} = \partial\Phi^*\left(-\frac{1}{\lambda}\nabla L(w)\right) \tag{7.9}$$

that is a natural subgradient generalization of the mirror-descent rule for Legendre regularization functions [28]:

[6] Slater's Condition, which in this case means $\Phi(\Delta w) < \epsilon$ for some vector $\Delta w \in \Re^d$

Algorithm 3 Boosted Coding

Input: Data matrix $X \in \Re^{m \times n}$, scalars $d, \lambda \in \Re^{+}$, convex functions $L(BW, X)$, $\Phi(W)$, and a function $b = \text{oracle}(-1/\lambda \nabla L(BW, X))$ which returns a new basis vector b corresponding to a non-zero row of Δw. (7.9).
Output: Active basis matrix $B \in \Re^{m \times d}$ and coefficients $W \in \Re^{d \times n}$.
Initialize: $W = 0^{d \times n}$, $B = 0^{m \times d}$.
for $t = 1$ to d **do**
  Add a new basis vector with zero weight:
  1: $B_t = \text{oracle}(-1/\lambda \nabla L(BW, X))$, $W^t = \vec{0}^T$
  Optimize W:
  2: $W = \arg\min_W L(BW, X) + \lambda \Phi(W)$
  if $\|W^t\|_2 = 0$ **then**
    return
  end if
end for

$$\Delta \hat{w} = \nabla \Phi^* \left(-\frac{1}{\lambda} \nabla L(w) \right). \tag{7.10}$$

By extending the mirror-descent rule to convex but non-differentiable regularization functions, we gain the ability to use mirror descent to optimize over infinite-dimensional spaces, while only ever storing a finite number of non-zero entries in w. The key is to employ regularization functions like L_1 that can have extremely sparse conjugate subgradients, and to find a computational trick (oracle) that can compute a subgradient (7.9) without ever explicitly computing the full gradient of the loss (which will be infinitely large for an infinite set of possible basis vectors). Note that sparsity is created by non-differentiability in the regularization function around its minimum. Hence a subgradient approach is necessary as we are primarily interested in a non-differentiable point of the regularization function.

In the following we show how a practical boosting algorithm can be constructed for the $L_{2,1}^2 + \gamma L_1^2$ regularization function used in the convex sparse coding objective function (7.1), by deriving the conjugate of the regularization function, Φ^*, computing a sparse subgradient over finite sets, and providing a tractable oracle heuristic for boosting from infinite sets of basis vectors (Algorithm 4). For this regularization function there are always subgradients consisting of a single new basis vector at each step, and we employ a step-wise fitting approach in Algorithm 3 to boost a basis matrix for sparse coding by adding one basis vector in each boosting step. Note that either ϵ or λ is assumed to be a known hyper-parameter. Here we assume λ is a known constant as this leads to greater deflation between boosting steps, and a smaller, less coherent[7] basis matrix.

7.2.3 The Regularization Conjugate $\Phi^*(Z)$

This section will prove the following lemma:

[7] The coherence parameter of a basis matrix is defined as the maximum absolute correlation between columns [111].

Lemma 4 *The Fenchel conjugate of:*

$$\Phi(W) = \frac{1}{2}\|W\|_{2,1}^2 + \frac{\gamma}{2}\|W\|_1^2$$

is the minimization over A of:

$$\Phi^*(Z) = \inf_A \frac{1}{2}\|Z - A\|_{2,\infty}^2 + \frac{1}{2\gamma}\|A\|_\infty^2 \tag{7.11}$$

The infimum over the additional variable A in (7.11) is known as the *infimal convolution* of the L_1^2 and $L_{2,1}^2$ terms. Since in the coding problem the dual variable Z is the negative loss gradient, $Z = -\frac{1}{\lambda}\nabla L(W)$, we will see in Section 7.3.3 that A has the effect of focusing the boosting step on a group of examples with high loss.

Lemma 4 follows from the the fact that the Fenchel conjugate of the sum of two functions is the *infimal convolution* of their conjugates [112]. The conjugate of the L_1^2 squared norm term is well known: the dual of a squared norm $f(x) = \frac{1}{2}\|x\|^2$ is simply the square of its *dual norm*. Let $\|\cdot\|$ be a norm on R^n The dual norm, $\|\cdot\|_*$ is defined as:

$$\|z\|_* = \sup_{\|x\| \leq 1} z^T x \tag{7.12}$$

For norms, the dual of the dual norm is the original norm $\|x\|_{**} = \|x\|$. The dual of the L_1-norm $\|x\|_1$ is the L_∞-norm $\|z\|_\infty$, and the dual of an L_p-norm $\|x\|_p$ is the L_q-norm $\|x\|_q$, where $1/p + 1/q = 1$ [29]. If $f(x)$ is a squared norm multiplied by a scalar γ, then the conjugate will be multiplied by γ^{-1}, $(\gamma f(x))^* = \gamma^{-1} f^*(z)$ [29]. Lemma 1 in Section 6.2 establishes that the conjugate of the $L_{2,1}^2$ term is $L_{2,\infty}^2$, and finishes the proof of Lemma 4.

7.2.4 The Subgradient $\partial_Z \Phi^*(Z)$

Applying the Fenchel subgradient boosting approach (Algorithm 3) to the sparse coding problem (7.1), requires computing a subgradient $\phi \in \partial_Z \Phi^*(Z)$ of (7.11), where the dual variable Z is the negative gradient of the loss: $Z = -\frac{1}{\lambda}\nabla L(W)$. Fortunately $\partial_Z \Phi^*(Z)$ is generally very sparse, and equals:

$$\begin{aligned}
\frac{\partial \Phi^*(Z)}{\partial Z_j^m} &= \begin{cases} 0 & \text{if } |Z_j^m| < \alpha \\ 0 & \text{if } \||Z^m| - \alpha\|_2^2 < \kappa \\ Z_j^m - \text{sign}(Z_j^m)\alpha & \text{otherwise} \end{cases} \\
\hat{\alpha} &= \|\hat{A}\|_\infty \\
\kappa &= \max_i \sum_k \left((|Z_k^i| - \hat{\alpha})_+\right)^2
\end{aligned} \tag{7.13}$$

where $\hat{\alpha}$ is the magnitude of the largest element of the matrix $|\hat{A}|$, which minimizes the infimal convolution in (7.11), and κ is equal to the maximal squared L_2 norm of any row in the matrix $Z - \hat{A}$. Deriving this subgradient requires analyzing some details of the infimal convolution, but is necessary for understanding the sub-problem involved in boosting from an infinite set of basis vectors.

Solving the Infimal Convolution

The solution $\hat{A}$ to the infimal convolution in (7.11) is:

$$\hat{A}^i_j = \begin{cases} Z^i_j & |Z^i_j| \leq \alpha \\ \text{sign}(Z^i_j)\alpha & |Z^i_j| > \alpha \end{cases} \tag{7.14}$$

where $\alpha = \|\hat{A}\|_\infty$. The infimal convolution is effectively a one-dimensional search over $\alpha \in [0, \|Z\|_\infty]$ which seeks to minimize the max of a set of piecewise quadratic functions ($\|Z^i - A^i\|_2^2, \ \forall i$). Replacing the $\|Z - A\|_{2,\infty}^2$ term in the infimal convolution with $\max_i \sum_j \left(\left(|Z^i_j| - \alpha\right)_+\right)^2$ produces:

$$\Phi^*(Z) = \min_\alpha \max_i \frac{1}{2\gamma}\alpha^2 + \frac{1}{2}\sum_j \left(\left(|Z^i_j| - \alpha\right)_+\right)^2 \tag{7.15}$$

Note that this optimization problem is convex with respect to α, since it is a max over convex functions. The minimizer, $\hat{\alpha}$, can be found to any desired accuracy by an interval bisection search[8] over $\alpha \in [0, \|Z\|_\infty^2]$. The subgradient (7.13) is the partial derivative[9] of (7.15).

7.3 Oracles for Infinite Bases

The convex exemplar-based approach to clustering formulated by [108] can be applied to sparse coding to create a simple oracle from a set of exemplars. In this case, the finite set of exemplars is the set of possible basis vectors, and the subgradient (7.13) derived above can be found efficiently by solving the infimal convolution. However, this solution can be improved by Algorithm 4, which optimizes over an infinite set of possible basis vectors. This section defines the non-convex optimization that must be solved to optimize over an infinite set of possible basis vectors. We start by considering two simpler limiting cases of $L_{2,1} + \gamma L_1$ regularization, L_1 regularization and $L_{2,1}$ regularization. We then present Algorithm 4 as a heuristic for finding a good solution in the general case.

7.3.1 L_1 Regularization

Consider the case where the loss function is the squared reconstruction error, $L(W, B) = \frac{1}{2}\|BW - X\|_F^2$, and the regularization function is the L_1-norm $\Phi(W) = \frac{1}{2}\|W\|_1^2$, with conjugate $\Phi^*(Z) = \frac{1}{2}\|Z\|_\infty^2$. In this case, a subgradient of the regularization conjugate is a matrix with one non-zero element, corresponding to a maximal element of $|Z|$. Therefore, the best possible basis vector (with unit L_2 norm) can be found by solving for the basis vector b_m that produces the largest element of Z. Since

[8] For a fixed level of accuracy the computational complexity of this search is at most O(Nd), where N is the number of examples (columns of Z) and d is the number of rows of Z.

[9] One might expect that since α is a function of Z^m_l, we should be interested in the total derivative: $\frac{\mathrm{d}\Phi^*(Z)}{\mathrm{d}Z^m_j} = \frac{\partial\Phi^*(Z)}{\partial Z^m_j} + \frac{\partial\Phi^*(Z)}{\partial\alpha}\frac{\partial\alpha}{\partial Z^m_j}$. However, since $\frac{\partial\Phi^*(Z)}{\partial\alpha} = 0$, the total derivative is equal to the partial derivative.

$Z = -\frac{1}{\lambda}\nabla L(W) = -\frac{1}{\lambda}B^T(BW - X)$, b_m is given by:

$$\begin{aligned} b_m &= \frac{E_m}{||E_m||_2}, \text{ where:} && (7.16)\\ E &= BW - X && (7.17)\\ m &= \arg\max_j ||E_j||_2^2. \end{aligned}$$

E is the reconstruction loss matrix, and m is the index of the example with the largest reconstruction loss. Hence L_1^2 regularized boosting corresponds to adding the L_2 projection of the loss gradient on the highest loss example to the active basis at each step.

7.3.2 $L_{2,1}$ Regularization

If instead we regularize with the $L_{2,1}^2$ block norm $\Phi(W) = ||W||_{2,1}^2$, with conjugate $\Phi^*(Z) = ||Z||_{2,\infty}^2$, any row Z^m with maximal L_2 norm is a subgradient of $\Phi^*(Z)$. Hence we can optimize over all possible basis vectors $\vec{b}$, by finding the basis vector $\vec{b}_m$ best correlated with the loss gradients on all the examples (measured by L_2-norm):

$$\begin{aligned} b_m &= \arg\max_{||b||_2 \le 1} ||b^T(BW - X)||_2^2 \\ &= \arg\max_{||b||_2 \le 1} b^T E E^T b. && (7.18) \end{aligned}$$

Again E is the reconstruction error (7.17). Although (7.18) is not convex, it is well known that b_m is the eigenvector associated with the maximum eigenvalue of the matrix EE^T [29]. Running Algorithm 3 with this choice of Φ is very related to PCA, and can be interpreted as PCA with incomplete deflation.

7.3.3 $L_{2,1} + \gamma L_1$ Regularization

The $L_{2,1} + \gamma L_1$ regularization used in (7.1) interpolates between the behavior of L_1 and $L_{2,1}$ based on the value of γ. The optimal new basis vector b_m will maximize the infimal convolution:

$$b_m = \arg\max_{||b||_2 \le 1} \left\{ \min_\alpha \frac{\alpha^2}{\gamma} + \sum_j \left((|b^T E_j| - \alpha)_+ \right)^2 \right\} \quad (7.19)$$

The introduction of the minimization over α makes (7.19) significantly more difficult to solve than the $L_{2,1}$ case (7.18). However, we can solve the infimal convolution for a finite set of candidate basis vectors, and we have already seen the solution for the limiting cases of $\gamma \to 0$ and $\gamma \to \infty$. To review: as $\gamma \to 0$, $\hat{\alpha}$ will converge to 0, and b_m will be equal to the solution of the $L_{2,1}$ only case (7.18). When $\gamma \to \infty$, $\hat{\alpha}$ converges to $\max_j |b_m^T E_j|$, and $L_{2,1} + \gamma L_1$ regularization reduces to L_1 regularization. In this case[10], $b_m \propto E_j$, where $||E_j||_2 = \max_i ||E_i||_2$, i.e. b_m is guaranteed to be proportional to some column of the reconstruction error E.

[10]The L_2 projection of any column of E with maximal L_2 norm is a valid choice for b_m

Algorithm 4 $L_{2,1} + \gamma L_1$ Heuristic

Input: Scalars γ, η and N, reconstruction error matrix $E \in \mathfrak{R}^{m \times n}$ where $E = BW - X$.
Output: New basis vector b.
Initialization: Compute a matrix $\tilde{B} \in \mathfrak{R}^{m \times N+1}$ containing candidate basis vectors as columns:
1: Set $\tilde{B}_1$ to the solution for $L_{2,1}$ regularization (7.18).
2: Set $\tilde{B}_2$ through $\tilde{B}_N + 1$ equal to the L_2 projections of the largest (in L_2 norm) N columns of E:

$$\tilde{B}_i = E_i / \|E_i\|_2, \ \forall \|E_i\|_2 > C$$

3: Compute the dual variable: $Z = -\frac{1}{\lambda} \tilde{B}^T E$
4: Find the index m of a maximal row by solving the infimal convolution:

$$m = \arg\max_i \min_\alpha \frac{1}{2\gamma}\alpha^2 + \frac{1}{2} \sum_j \left(\left(|Z_j^i| - \alpha \right)_+ \right)^2$$

5: Assign b to the best candidate: $b = \tilde{B}_m$.
6: Improve b by gradient ascent using (7.20):
repeat
7: $z = -\frac{1}{\lambda} b^T E$
8: $b = b + \eta \frac{\partial \Phi^*(z)}{\partial b}$
until $\| \frac{\partial \Phi^*(z)}{\partial b} \|_2 < \epsilon$
Return b

Algorithm 4 provides an empirically effective method for estimating β and b_m. It starts by combining the solutions for the $L_{2,1}$ and L_1 regularization cases discussed above into a matrix of candidate basis vectors $\tilde{B}$. Then a promising choice for b_m is selected by finding the basis vector in $\tilde{B}$ which produces a maximal row of the infimal convolution over $Z = -\frac{1}{\lambda}\tilde{B}^T E$. Finally b_m is improved by gradient ascent to maximize the conjugate of the regularization, $\Phi^*(Z^m)$, where $Z^m = -\frac{1}{\lambda}b_m^T E$ and the gradient is:

$$\begin{aligned} \frac{\partial \Phi^*(Z^m)}{\partial b_m} &= \frac{\partial \Phi^*(Z^m)}{\partial Z^m}\frac{\partial Z^m}{\partial b_m} \\ &= -\frac{1}{\lambda}\sum_j E_j \text{sign}(Z_j^m)(|Z_j^m| - \alpha)_+ \\ \text{where:} \qquad \alpha &= \min_\alpha \frac{\alpha^2}{\gamma} + \sum_j \left((|b^T E_j| - \alpha)_+\right)^2 \end{aligned} \tag{7.20}$$

b_m can be interpreted as the solution to an eigenvalue problem similar to (7.18), except for a weight on the loss of each example E_j (Figure 7.2) introduced by the infimal convolution.

Lemma 5 *There exists a diagonal matrix β, such that* (7.19) *can be written as:*

$$b_m = \arg\max_{\|b\|_2 \le 1} b^T E \beta^2 E^T b \tag{7.21}$$

To prove this lemma, define $\hat{\alpha}$ to be the value of α which minimizes (7.19) for $b = b_m$. Rewrite the maximization over b as:

$$\begin{aligned} &\max_{\|b\|_2 \le 1} \left\{ \min_\alpha \frac{\alpha^2}{\gamma} + \sum_j \left((|b^T E_j| - \alpha)_+\right)^2 \right\} \\ &= \frac{\hat{\alpha}^2}{\gamma} + \sum_j \left(|b_m^T E_j| \frac{(|b_m^T E_j| - \hat{\alpha})_+}{|b_m^T E_j|} \right)^2 \\ &= \frac{\hat{\alpha}^2}{\gamma} + \sum_j \left(|b_m^T E_j| \beta_j^j \right)^2 \\ &= \frac{\hat{\alpha}^2}{\gamma} + \max_{\|b\|_2 \le 1} b^T E \beta^2 E^T b \\ &\qquad \text{where: } \beta_j^j = \frac{(|b_m^T E_j| - \hat{\alpha})_+}{|b_m^T E_j|} \end{aligned} \tag{7.22}$$

The flexibility of the $L_{2,1} + \gamma L_1$ regularization combination is evident on the MNIST-ROT dataset of rotated handwritten digits [12]. As shown in Figure 7.3, for large values of γ when the L_1 penalty term dominates, the boosted coding algorithm learns basis vectors that look like particular digits at particular orientations. An intermediate value of γ balances the two terms and produces a basis that looks like strokes. Small values of γ produce a PCA-like solution.

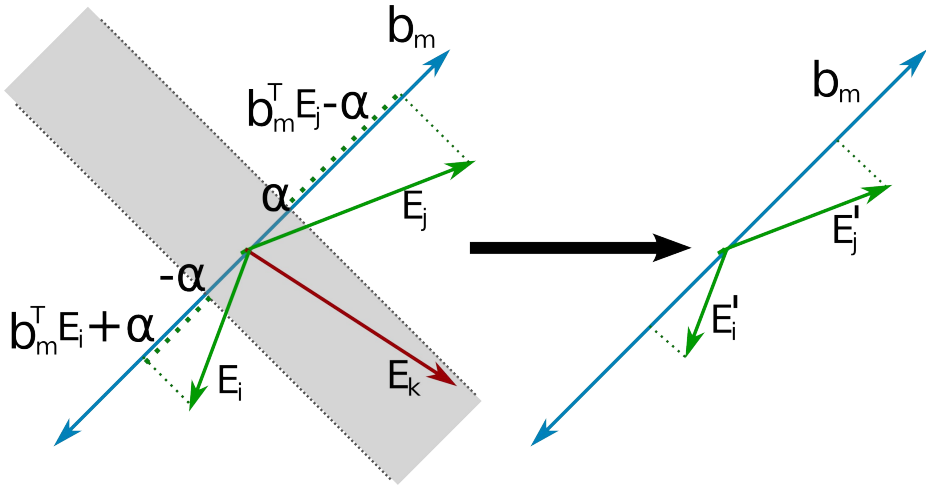


Figure 7.2: The vector b_m which maximizes the infimal convolution (left) for squared loss, is the result of maximizing $\sum_i \left((|b_m^T E_i| - \hat{\alpha})_+\right)^2$. There exists a diagonal weight matrix β that reduces this problem to a standard eigenvalue problem (right) by rescaling the reconstruction loss vectors for each input example: $E' = E\beta$. In the diagram above, after rescaling example k has no weight, and b_m is computed to maximize correlation with the rescaled loss gradients on examples i, and j.

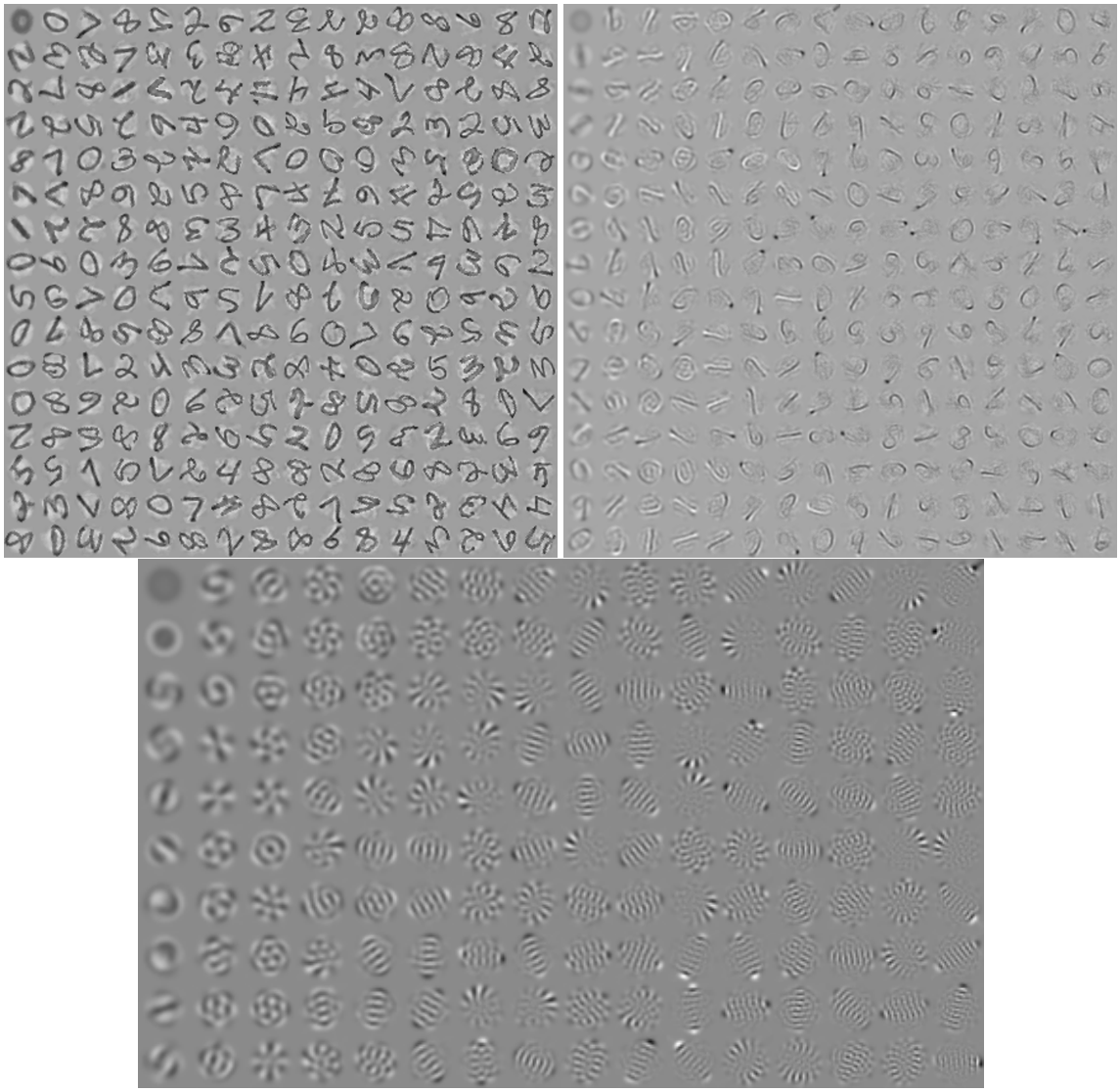


Figure 7.3: Changing the value of γ smoothly interpolates between learning an example basis and a PCA basis as shown on the MNIST-rotated dataset above. Top left: $\gamma = 1$ produces basis vectors that are similar to individual input examples. An intermediate value of $\gamma = 10^{-2}$ (top right) balances the influence of both terms and produces a stroke-like basis. A small value of $\gamma = 10^{-}3$ approaches the PCA solution (bottom).

7.4 Results on Image Denoising

Boosted coding (Algorithm 3) was applied to the task of image denoising in order to evaluate its performance on a real-world task that is well-suited to sparse coding [98], and lends itself particularly well to visualizing the behavior of the algorithm. The performance of alternating optimization[11] and boosted coding turn out to be quite similar on this task, with a slight advantage for the boosted approach. This result provides reassuring evidence that the non-convex but simple alternating optimization algorithm is not seriously impaired by inferior local minima on this task.

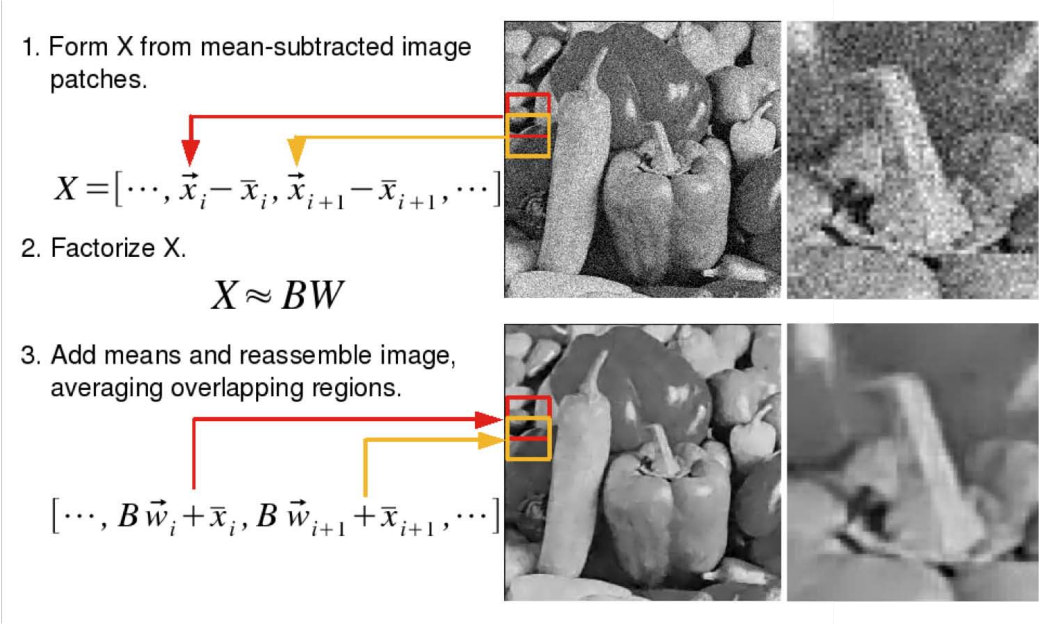


Figure 7.4: Image denoising proceeds by extracting overlapping patches from a noisy input image. Each patch is rearranged to form a column of the data matrix X. After X is approximated as B*W, the denoised image is reconstructed by averaging the overlapping patches.

The image denoising approach is modeled on the K-SVD algorithm presented in [98]. As shown in Figure 7.4, overlapping patches are extracted from a noisy input image. Each patch is rearranged into a vector $\vec{x}_i$, the mean $\bar{x}_i$ of the patch is subtracted, and the result becomes a column of the data matrix X. X is factorized into the product of B and W using sparse coding. Non-zero components of W are then refit without regularization. Finally, the denoised image is reconstructed by adding back the mean of each patch, and averaging areas of the image where multiple patches overlap. These experiments used 8x8 pixel patches with an overlap of four pixels between neighboring patches. The alternating optimization approach has two hyperparameters–the regularization constant λ and d, the number of columns of B–and boosted coding has two regularization constant hyperparameters λ and γ. The hyper-parameters of both algorithms were independently tuned for maximal performance, in order to isolate the effect of the different optimization strategies.

[11] Used in many past works such as [21].

Independent and identically distributed Gaussian noise ($\sigma = 0.1$) was added to five common benchmark images to match the assumed noise model of the L_2 loss function (Figure 7.5, left-most column). Subtracting the mean of each patch removes low-frequency components of the image, leaving the coding problem to focus on identifying which high-frequency components of the image are contained in each patch. If W is simply set to zero, the result is to average each 8x8 patch of the image, which improves the Signal-to-Noise Ratio (SNR) of the low-frequency components of the image at the expense of the high-frequency details in the image (Figure 7.5, second column).

Coding X with alternating optimization or the convex up-to-an oracle approach presented in Algorithm 3 (Figure 7.5, right side) restores high-frequency detail by finding basis vectors that describe shared patterns across all patches. In these experiments both algorithms produced roughly equivalent results in terms of SNR, with a slight advantage for the convex approach. The parameters of both algorithms were tuned for maximal performance, in order to isolate the effect of the different optimization strategies.

7.4.1 Boosted Coding

The effect of relaxing the non-convex rank constraint on W by substituting $L_{2,1}$ regularization can be clearly seen by comparing the basis vectors selected by the two algorithms. Boosted coding selects the most important basis vectors first, and those are used by many image patches due to the "group discount" provided by the L_2 norm in $L_{2,1}$ regularization. This causes the signal to noise ratio to rise quickly at the beginning of the process and then level off once most of the underlying signal can be represented by the basis vectors. As shown in Figure 7.6, images with just a few dominant high-frequency patterns (such as "house" and "boat") converge quickly, while images with a variety of repetitive texture require more steps.

The first basis vectors chosen are also smoother in appearance than basis vectors chosen at later steps, as can be observed in the bases learned for each image Figure 7.7. This follows from the fact (shown in Section 7.3.3) that each step of boosting with $L_{2,1} + \gamma L_1$ regularization will find a basis vector that is maximally correlated with the reconstruction error on a subset of the image patches (Figure 7.8). Therefore, in later rounds of boosting much of the structure of the image patches is already explained, and the reconstruction error on each patch will consist largely of noise.

The basis selected by boosted coding is also less coherent (i.e. the basis vectors are less correlated with each other) than the basis selected by alternating optimization as shown in Table 7.1. Basis coherency is useful when denoising with only L_1 regularization, as the image structure lies mainly in a low-dimensional subspace of of the 64-dimensional space of 8x8 pixel patches, while the noise is spread uniformly throughout the space. However, because of the group discount on heavily used features afforded by $L_{2,1}$ regularization, Boosted Coding is able to perform well with a less coherent basis, as basis vectors that are used by fewer patches in the image are penalized more heavily.

7.4.2 Alternating Optimization

A common objection to the traditional, alternating optimization approach to sparse coding (5.5) is that the non-convex rank constraint on B could results in the algorithm returning inferior local minima. Anecdotal evidence suggests this problem should be most acute for relatively small basis

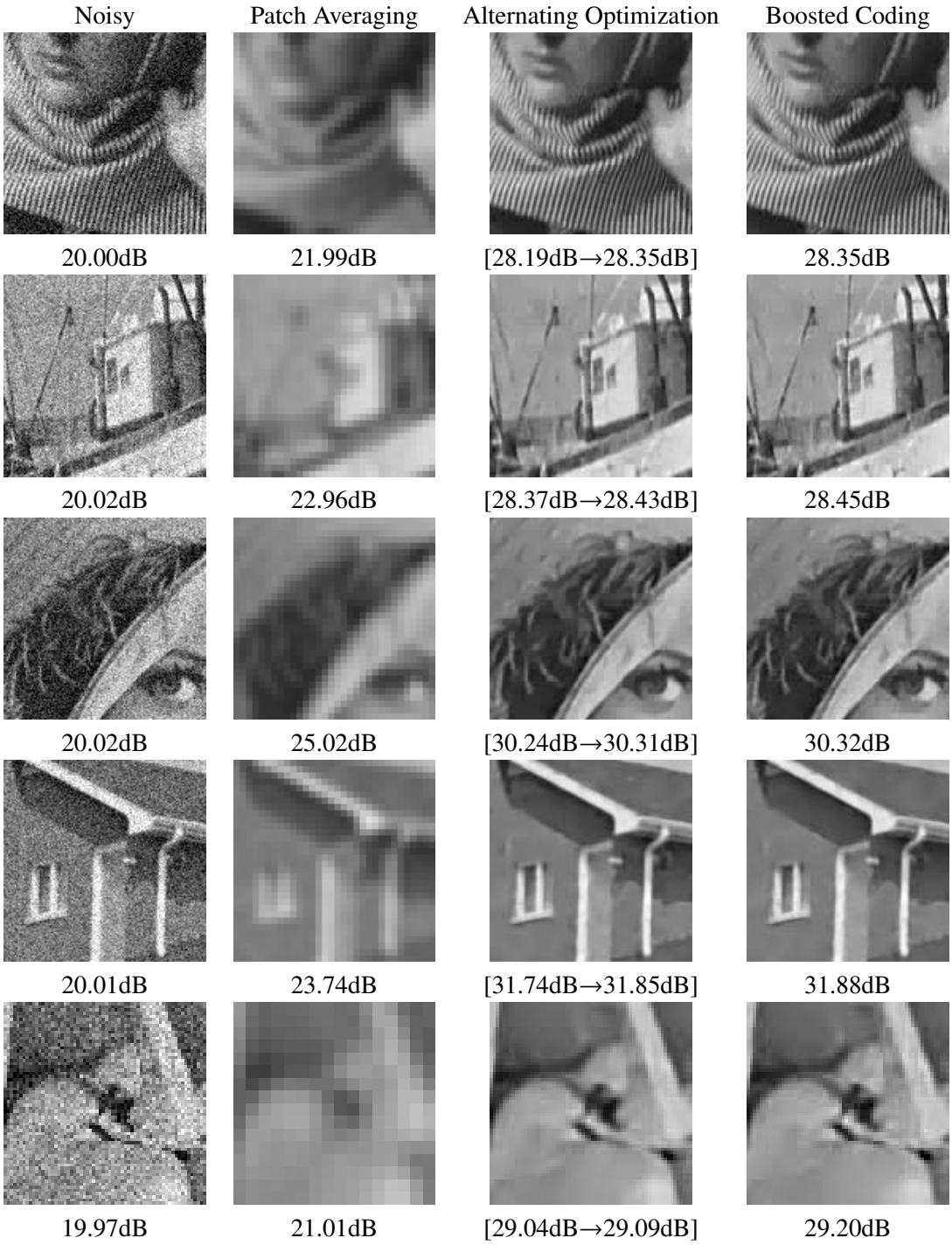


Figure 7.5: Results on five benchmark images show that both alternating optimization and boosted coding produce similar results for image denoising. Shown is a closeup of the center portion of each image. The range reported for alternating optimization is the best and worst performance from 20 randomly initialized trials.

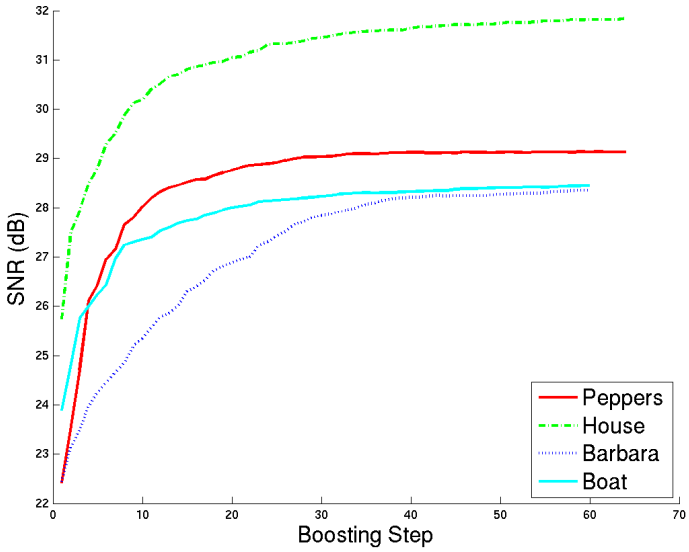


Figure 7.6: Basis vectors selected by the oracle quickly maximize denoising performance. Images with fewer regular patterns like House and Boat require fewer steps than complicated images like Barbara.

	Correlation between basis vectors	
Image	Boosted Coding	Alternating Optimization
Barbara	0.57 (0.12)	0.92 (0.16)
Boat	0.55 (0.21)	0.88 (0.30)
Lena	0.66 (0.24)	0.90 (0.35)
House	0.63 (0.23)	0.93 (0.33)
Peppers	0.60 (0.21)	0.91 (0.33)

Table 7.1: Alternating optimization tends to produce very coherent (correlated) basis vectors. Because of the additional $L_{2,1}$ regularization, boosted coding performs just as well with a less coherent basis. Shown is the maximum absolute correlation between basis vectors for the best-performing basis on each image, with the mean correlation given in parentheses.

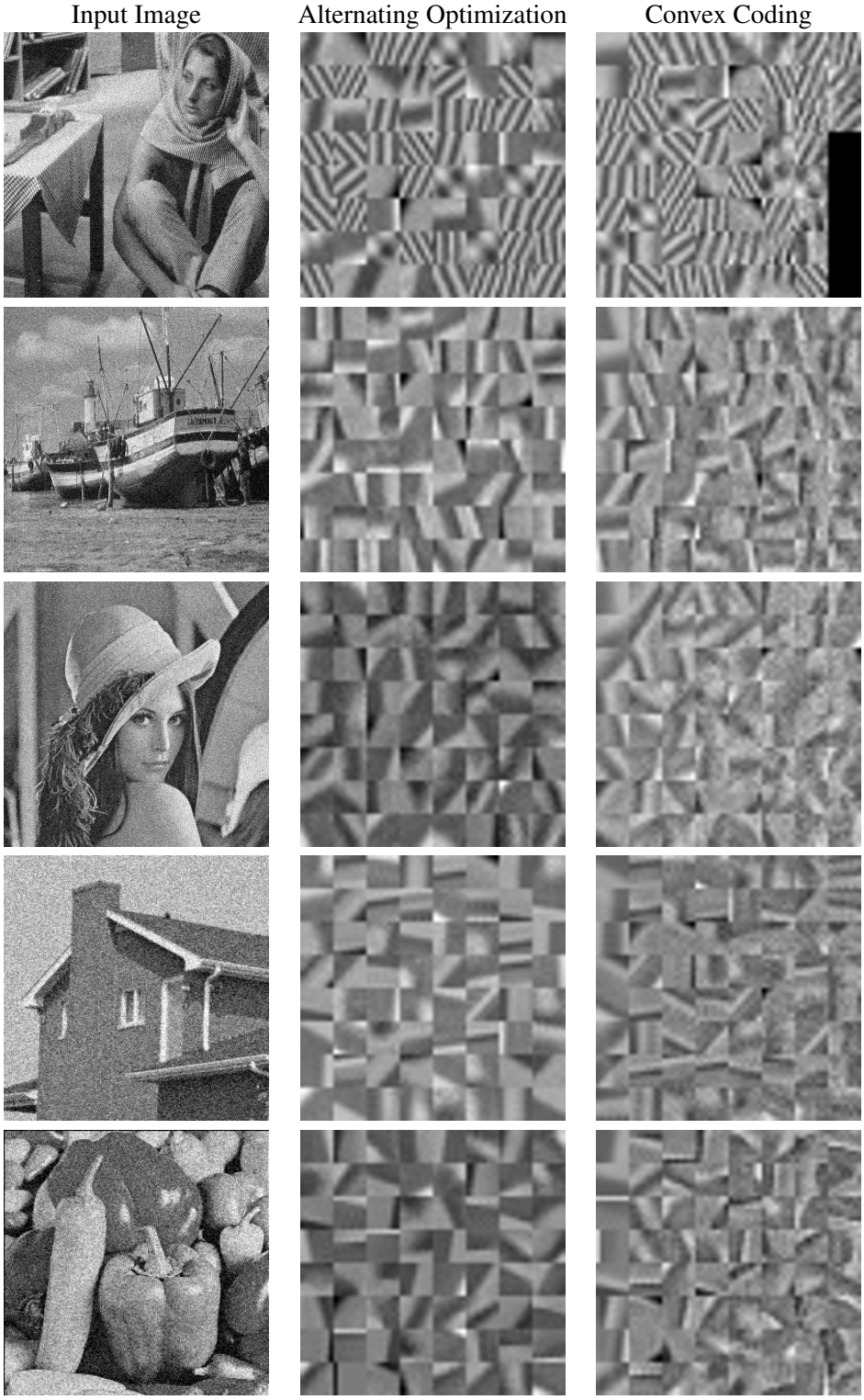


Figure 7.7: **Left:** Noisy input image. **Center:** 8x8 basis vectors learned by the alternating optimization and boosted coding algorithms. The boosted coding basis vectors are displayed in the order they were selected (top to bottom and left to right). In some cases boosted coding chose less than 64 basis vectors.

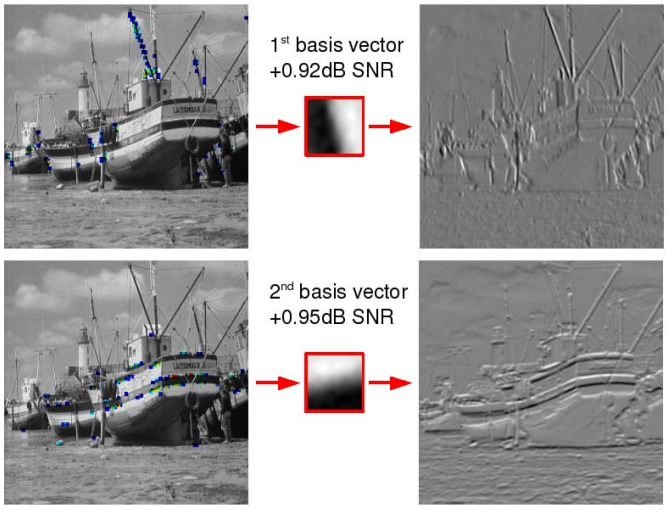


Figure 7.8: The basis vector selected by the oracle is chosen to maximize its correlation with a weighted set of high-error patches. **Left:** patches used by the oracle to compute the new basis vector (**center**) are colored according to weight, where blue is low and red is high. **Right:** filtering the image with the new basis vector indicates where it is most useful.

sizes, where there are only a small number of randomly-initialized basis vectors. In this case there are fewer degrees of freedom available to let alternating optimization escape a local minima. In our experience, inferior local minima, while they do occur, have a relatively small effect on the performance of the alternating optimization algorithm. Table 7.2 quantifies this assertion by showing the results of repeatedly running the alternating optimization image denoising algorithm from different random initializations of the basis vectors. The optimization alternated between B and W 20 times. This represents a stress-test for the alternating optimization algorithm as the basis contains only eight basis vectors. Even in this challenging case alternating optimization performs reasonably consistently, although Figure 7.1 provides a detailed look at a case where one local minima was superior to the others.

	SNR Improvement (dB)		
Image	Min	Max	Mean
Lena	4.60	4.78	4.73 ± 0.06dB
Barbara	3.10	3.38	3.20 ± 0.11dB
Boat	4.57	4.65	4.62 ± 0.03dB
Peppers	6.55	6.84	6.71 ± 0.1dB
House	6.20	6.25	6.22 ± 0.02dB

Table 7.2: Variance observed when using alternating optimization L_1-regularized sparse coding to denoise grayscale images. Shown is the mean and standard deviation of the improvement (vs. patch averaging) in then Signal-to-Noise ratio of the image observed during 5 random initializations of B.

Chapter 8

Learning Through Planning Modules

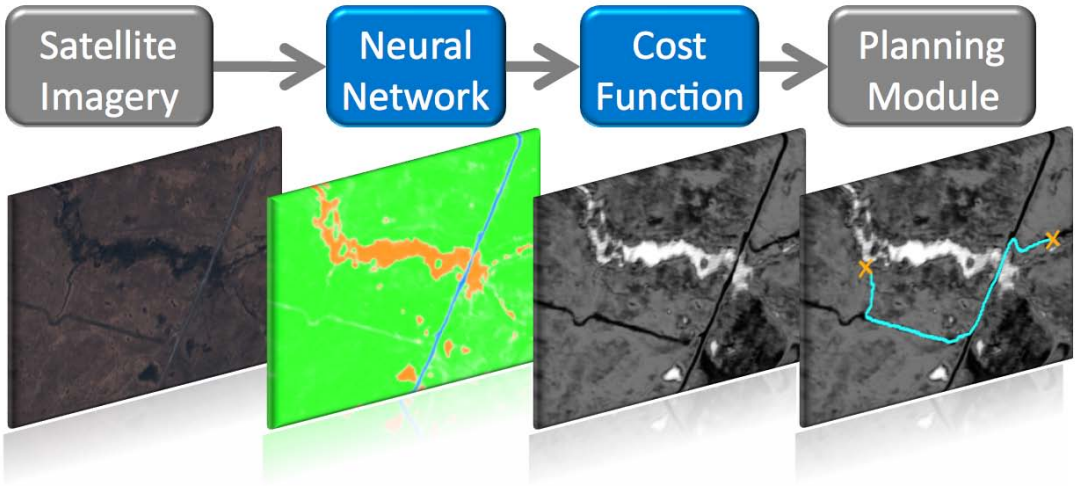


Figure 8.1: Satellite imagery is a valuable tool for finding paths through complex environments. The system for navigation from satellite imagery used in this chapter uses a neural network to classify each patch of the image, and then predicts a "cost" for traversing the cell at the center of the patch. A planning module then uses the A^* algorithm [49] to find a minimum cost path from a start to a goal location. However, the use of a planning algorithm creates challenges for using learning to improve previous modules in the system.

As discussed in Chapter 3, many robotic systems incorporate planning modules to generate structured outputs. For instance, the navigation system shown in Figure 8.1 uses the A^* planning algorithm to efficiently find a path from the robot's current location to a goal location that optimize a cost function. Other applications where planning modules include everything from manipulation [24], to predicting the future path of people or other tracked targets [113], and even automated restaurant management [114].

Planning modules and structured outputs such as paths can present serious challenges for learning. It is often hard to construct effective loss functions for structured outputs, and planning modules

are not differentiable, as they compute an arg min or arg max operation to produce the structured output. Without a way to back-propagate a loss gradient — or even to specify a loss function — many robotics projects have been forced to depend heavily on human-guided parameter tuning to produce reasonable behavior [51].

Figure 8.2: The Maximum Margin Planning (MMP) algorithm [5, 6] learns cost functions that make behavior demonstrated by experts optimal to the planning algorithm used by a robot. In the above image, an experienced driver is tele-operating a mobile robot and providing an example of the driver's preference for driving on roads. This and other demonstrations were used by MMP to learn a cost function for the local perception system of the robot which imitated the expert's behavior, and outperformed an engineered cost function that had been developed through extensive field testing and parameter tuning [23, 26].

Recently, Ratliff et al. [5] introduced the Maximum Margin Planning (MMP) algorithm which provides a useful loss function and subgradient descent optimization procedure for learning a cost function that causes a planning module imitate demonstrated expert behavior. For mobile robot navigation, this innovation meant that the cost function module (such as Figure 8.1) could now be learned directly from example paths drawn on a map, or from teleoperation of the robot (Figure 8.2) as in my previous work [23]. A version of MMP that could learn non-linear cost functions was introduced in [23, 6], and found to be effective during extensive field testing on a wide variety of problems [24, 22, 25, 26].

In this chapter, techniques are developed and demonstrated for employing MMP in the global coordination of deep modular systems through planning modules. Previous work used the input features to the cost function as given. Here the emphasis is on adapting those inputs through subgradient descent. Additionally, the pre-training paradigm developed for learning deep neural networks [4, 3], and discussed in Section 2.2 is applied by first pre-trained the input features to the cost generation function on an "auxiliary" task, and then adapting the input features for the "target" task of the system through gradient descent on the target task loss function. This work demonstrates the usefulness of subgradient backpropagation through non-differentiable planning modules on the ap-

plication shown in Figure 8.1. Additionally, a modified version of the loss function used by MMP is developed, which corrects undesirable behavior that was observed early in the optimization process.

The next section provides an overview of the Maximum Margin Planning (MMP) algorithm for learning through planning modules. Section 8.2 discusses a failure mode encountered when applying backpropagation to standard MMP, and formulates a modified objective function that avoids it. Finally, Section 8.3 provides an experimental demonstration of the approach, and of the benefits of pre-training intermediate modules on relevant tasks.

8.1 Maximum Margin Planning

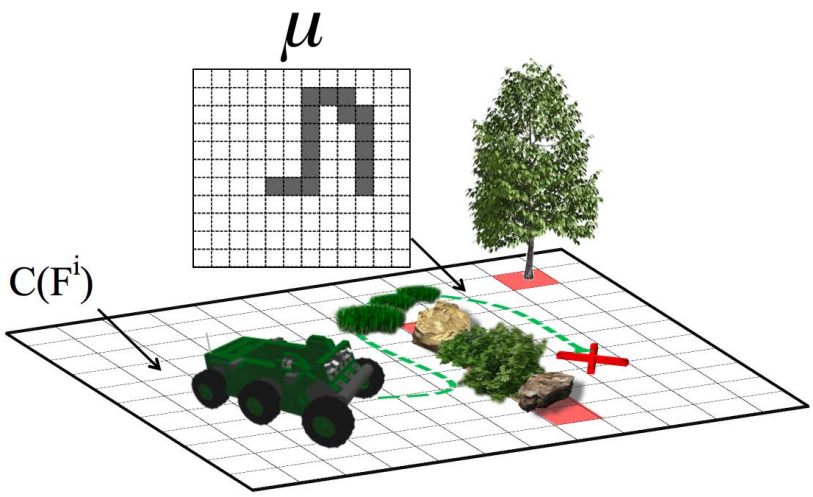


Figure 8.3: Learning a cost function C(F) for mobile robot navigation with Maximum Margin Planning (MMP). The cost function, represented as a "cost map" of the environment in the 2-D case, defines the negative reward for traversing each cell i of the grid above by computing a function $C(F^i)$ of the i^{th} row of the input feature matrix F. The planner returns a path μ that contains the distance the robot would traverse through each cell as it travels to the goal.

Maximum Margin Planning (MMP) [5, 6] is a key technique for for learning through a planning module. MMP is applicable to general Markov Decision Processes (MDPs) but for concreteness here the algorithm will be discussed as it is applied to the mobile robot navigation problem. Figure 8.3 shows the problem setup. A mobile robot is tasked with driving from its current location to a goal location indicated by a red X. The environment around the robot is discretized into a 2-D grid containing m cells, and the robot has access to a matrix of features $F \in \Re^{mxd}$ containing one row F^i for every cell i in the 2-D grid. A cost function $C(F^i)$ returns the "cost" (negative reward) to the robot of traversing cell i of the grid, and the vector $C(F) \in \Re^m$ is a "cost map" containing the cost of every grid cell. A plan to the goal is represented by a vector $\mu \in \Re^m$ that records the distance the robot will traverse through each cell of the cost map. Costs are assumed to be additive, and the total cost of a path is $C(F)^T\mu$. The planning algorithm is assumed to return the path $\hat{\mu}$ with the minimum

total cost:

$$\hat{\mu} = \arg\min_{\mu} C(F)^T \mu. \tag{8.1}$$

The MMP algorithm takes as input a set of N example paths $\mu_1, ...\mu_N$ and attempts to construct a cost function that will make the output of the planner match the examples, with a margin. Formally this can be written as a set of constraints:

$$C(F)^T \mu_j \leq (C(F) - l_j)^T \hat{\mu}_j, \quad \forall j \in \{1, ..., N\}, \tag{8.2}$$

where l_j is a loss function that adds a margin requirement by penalizing states on the example path. Additionally, a regularization penalty $\Phi(C)$ is applied to the cost function and together these produce the MMP objective function:

$$\mathfrak{L} = \frac{1}{N} \sum_{j=1}^{N} \left(C(F)^T \mu_j - \min_{\mu \in G_j} (C(F) - l_j)^T \mu \right) + \lambda \Phi(C). \tag{8.3}$$

Early work on MMP [5, 23] used a linear cost function, $C(F) = Fw$, with L_2-norm regularization on the parameter vector w. This choice of cost function makes the objective function (8.3) convex, and it can be optimized with subgradient descent [115] by following the subgradient:

$$\frac{\partial \mathfrak{L}}{\partial w} = \frac{1}{N} \sum_{j=1}^{N} F(\mu_j - \hat{\mu}_j)^T, \quad \hat{\mu_j} = \arg\min_{\mu \in G_j} (e^{Fw} - l_j)^T \mu. \tag{8.4}$$

Note that this subgradient is merely the difference in the sum of the feature vectors along the example paths and the minimum cost paths. Care must be taken when using a linear cost function with planning algorithms like A^*, which require that all cells must have positive costs, as the positivity requirement restricts the set of valid w and requires a projection step.

An alternative approach that is empirically more effective employs *exponentiated functional gradient descent* (EFGD) to ensure that all costs will be strictly positive without requiring a projection step [6, 22]. An excellent derivation of EFGD is given in [6], here we note that the MMP functional gradient is a sum of delta functions at locations where the example and minimum cost paths differ, weighted by the cost function evaluated at those locations. This functional gradient is then projected on a "direction set" of functions and the most correlated function in the direction set is exponentiated and multiplied by the current cost function. If the direction set is composed of all linear regression functions on the input, $h(x) = x^T w$, then the linear regression function most correlated with the functional gradient can be found by solving the weighted classification problem:

$$\hat{w} = \arg\min_{w} \sum_{i=1}^{m} \alpha_i (y_i - F^i w)^2, \tag{8.5}$$

where $\alpha_i = |C(F^i) \sum_{j=1}^{N} (\mu_j^i - \hat{\mu}_j^i)|$ and $y_i = \text{sign}(\sum_{j=1}^{N} (\mu_j^i - \hat{\mu}_j^i))$. When the most correlated linear regressor $x^t \hat{w}$ is exponentiated and multiplied by the previous cost function it builds a cost function of the form:

$$C(F) = c_0 + e^{Fw}, \tag{8.6}$$

which is guaranteed to have strictly positive costs. Many planning algorithms, including variants of A^*, add a small positive scalar constant c_0 for computational efficiency reasons.

Here we seek to jointly optimize all of the modules in a system, by leveraging modular learning approaches [1, 2] that backpropagate gradients from a loss function on the final system output. Our goal is to use the loss function over paths defined by MMP as a source of loss gradients for this modular learning approach. As the cost function constructed by EFGD with linear regressors has proven to be successful in extensive empirical testing [22], we use the effective functional form of the cost function constructed by EFGD with linear regressors (8.6) to construct a parametric objective function that captures the same intuition as MMP:

$$\mathfrak{L} = \frac{1}{N}\sum_{j=1}^{n}\left(\left(\vec{c_0} + e^{Fw}\right)^T \mu_j - \min_{\mu \in G_j}(\vec{c_0} + e^{Fw} - l_j)^T \mu\right) + \frac{\lambda}{2}\|w\|_2. \tag{8.7}$$

8.2 MMP Subgradient Backpropagation

Previous work on MMP has focused on learning a cost function for a given set of input features, or adding additional input features through functional gradient descent. This work develops methods for modifying existing input features with gradient backpropagation. If the input feature matrix F in (8.7) is produced by a differentiable function $F(X, \theta)$ with parameter vector θ, subgradient descent can be applied to θ using a subgradient $\delta \in \frac{\partial \mathfrak{L}}{\partial \Theta}$ of the objective with respect to θ. Interestingly, as the objective function (8.7) is not convex, gradient descent with respect to either θ or the parameters of the cost function w can produce undesirable results. To see why it is helpful to analyze a subgradient of (8.7) with respect to w, $\frac{\partial \mathfrak{L}}{\partial w}$, which is the sum of the cost-weighted feature counts on locations where the example paths differs from the minimum cost paths:

$$\frac{\partial \mathfrak{L}}{\partial w} = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1:m} e^{F^i w} F^i(\mu_j^i - \hat{\mu}_j^i), \quad \hat{\mu_j} = \arg\min_{\mu \in G_j}(e^{Fw} - l_j)^T \mu. \tag{8.8}$$

8.2.1 Cost Normalization

When the example path μ_j is longer than the current minimum cost path, it can happen that the cost-weighted feature counts on every feature will be higher for the example path than for the minimum cost path, and gradient descent will decrease the weight on all features, pushing all costs toward their minimum value. If the number of features is small and the example paths are significantly longer than the current minimum cost paths, this situation can occur quite regularly. Following the subgradient of the objective with respect to the input feature matrix $F(X, \theta)$, $\frac{\partial \mathfrak{L}}{\partial F}$, can produce a similar result. The gradient may indicate that $\mathfrak{L}$ should be minimized by reducing the magnitude of every input feature, again driving the cost map toward a uniform cost.

To avoid this failure mode, the objective (8.7) was modified so that it would not be improved by changes that simply decrease all costs. The standard MMP formulation defines a set of constraints (8.2) that the desired cost function should satisfy. These constraints in turn define the MMP objective function and its subgradients. Therefore, to eliminate the uniform cost failure mode the constraints must be modified to so that the objective function gradient does not simply encourage

lowering costs. If the cost function is always positive and there is at least one example with non-zero path length, the total cost of all examples will be positive, $\sum_i C(F)^T \mu_i > 0$. Dividing the set of constraints (8.2) by the total example cost and rearranging produces the equivalent[1] constraints:

$$1 - \frac{(C(F) - l_j)^T \hat{\mu}_j}{\sum_j C(F)^T \mu_j} \leq 0, \quad \forall j \in \{1, ..., N\}. \tag{8.9}$$

Substituting these "normalized" constraints (8.9) into the MMP objective function (8.7) produces a novel normalized version of the MMP objective function:

$$\mathfrak{L}_{\text{norm}} = 1 - \frac{\sum_{j=1}^{n} \min_{\mu \in G_j} \left(\vec{1} + e^{Fw} - l_j\right)^T \mu}{\sum_{j=1}^{n} \left(\vec{1} + e^{Fw}\right)^T \mu_j} + \frac{\lambda}{2}\|w\|_2. \tag{8.10}$$

The normalized objective function $\mathfrak{L}_{\text{norm}}$ is not improved by changes that simply decrease all costs. Adding normalization means that $\mathfrak{L}_{\text{norm}}$ can only be improved by increasing the ratio of the cost of the plans returned by the planning algorithm divided by the cost of the example paths. The subgradients of $\mathfrak{L}_{\text{norm}}$ reflect this new focus:

$$\frac{\partial \mathfrak{L}_{\text{norm}}}{\partial w_k} = \frac{C_{\hat{\mu}}}{C_\mu} \sum_{j=1}^{N} \sum_{i=1:m} e^{F^i w} \left(\frac{\mu_j^i}{C_\mu} - \frac{\hat{\mu}_j^i}{C_{\hat{\mu}}}\right) F_k^i \tag{8.11}$$

$$\frac{\partial \mathfrak{L}_{\text{norm}}}{\partial F_k^i} = \frac{C_{\hat{\mu}}}{C_\mu} \sum_{j=1}^{N} e^{F^i w} \left(\frac{\mu_j^i}{C_\mu} - \frac{\hat{\mu}_j^i}{C_{\hat{\mu}}}\right) w_k \tag{8.12}$$

$$\text{where:} \quad C_\mu = \sum_j C(F)^T \mu_j, \quad C_{\hat{\mu}} = \sum_j C(F)^T \hat{\mu}_j \tag{8.13}$$

8.3 Experiments

This section provides experimental validation for employing backpropagation through a planning module, and demonstrates the value of pre-training intermediate modules. However, providing an accurate comparison required an experimental setup that eliminated several complicating factors encountered in many real applications.

8.3.1 Complicating Factors

MMP is often used to imitate human behaviors with an efficient planning algorithm, a pairing that causes multiple inconsistencies. First, humans use planning strategies that are both high-dimensional and generally sub-optimal. For instance, experienced drivers account for nonholonomic motion constraints, incorporate complicated maneuvers like 3-point turns, and anticipate possible velocity and orientation dependent traction constraints. Additionally, human driving is only approximately optimal, and drivers often prefer plans which require fewer or less precise maneuvers to optimal plans which may be harder to envision and execute. Hence efficient, grid-based

[1] These constraints are equivalent in the sense that they share minima.

planning algorithms such as A^* are generally unable to explain human driving behavior as optimal plans with respect to an underlying cost function. In practice various strategies, such as re-planning the example paths within a corridor around the original path, have been developed to reduce the problems these inconsistencies cause for MMP [22].

Robotic systems also often use efficient planning algorithm implementations which can introduce quantization and range non-linearities into the cost function. The field D* [116] implementation used in [23, 22, 25] represents costs as 12-bit integers, with the result that costs near the minimum cost are subject to significant quantization, and the highest cost cell is at most 4096 times as expensive as the lowest cost cell. The practical effect of a limited cost range is that unless the planner is restricted from ever including cells with the highest representable cost, the robot will at some point choose to drive through a brick wall instead of driving a longer route on an open road. For MMP learning this means that examples that show a long detour through a medium cost field in order to avoid a narrow but impassable ditch may not be feasible.

8.3.2 Experimental Setup

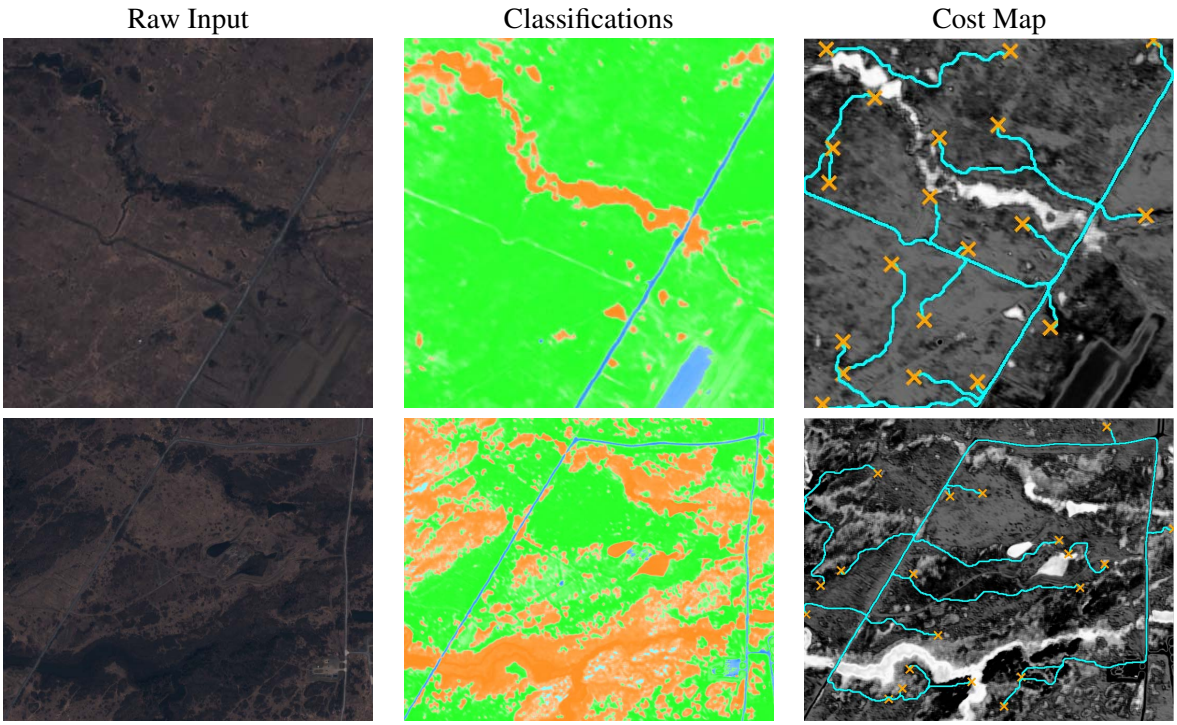


Figure 8.4: Validation (top) and training (bottom) areas constructed by learning a 5-class classification task (center column) on 5x5 pixel patches of the satellite images (left column). The classifications are used by an exponential cost function to create cost maps (right column) that produce reasonable driving behavior as shown by the minimum cost paths (cyan lines) between randomly sampled locations (orange Xs).

The goal of these experiments is to evaluate the effects of MMP subgradient backpropagation and module pre-training, with both human-designed and sparse coding auxiliary tasks, in learning modular robotic systems. Consequently, the setup for these experiments was carefully engineered to eliminate the complicating factors described previously. To eliminate cost range and quantization problems, a double precision version of A^* was employed for path planning. Generating a set of feasible example paths required some care. First, a set of five terrain classes[2] were labeled on a section of overhead satellite imagery that contained 2.4m square pixels with 8-bit values for each of four color channels (red, green blue, and near-infrared). Then a neural network with one hidden layer of size 30 was trained to predict the terrain classes given 5x5 pixel patches of the input. The neural network was then evaluated on the entire section of satellite imagery, and a bias feature was appended to the predicted output for each cell to form a feature matrix F. Next a cost function of the form $C(F, w) = 1 + e^{Fw}$ was learned by iterating the construction of example paths given a costmap, with the learning of w given a fixed set of example paths. Given a fixed cost map, start, goal, and select intermediate points were selected for each example path, and A^* was used to find a minimum cost path between the selected points. Then, the cost function parameter vector w was updated using MMP on the set of example paths. This process was iterated until the example paths exhibited the desired behavior without requiring intermediate constraint points.

The learned neural network and cost function were then used to create "ground truth" cost maps for physically distinct test and training regions of satellite imagery which contained similar terrain. All of the terrain examples used for learning the ground truth neural network were drawn from the test regions, and a separate six-category[3] terrain classification task was defined by labeling patches in the training region for use as a pre-training auxiliary task. Another "automatic" auxiliary task was generated by sparse coding 30,000 randomly selected patches from the training map. As the training paths use features like roads which only cover a small portion of the training image, half of the randomly selected patches were sampled from areas on the training paths. The fast SPAMS L_1 sparse coding implementation of [117] was used to sparse code the paths using parameters determined by optimizing a pseudo-objective (Figure 8.5). After the patches had been sparse coded, a 100 hidden unit neural network with linear output units was trained to approximate the sparse codes produced for the patches. Figure 8.4 shows the input satellite imagery, the five-category neural network classifications, and the cost map used as ground truth for each region. Ten example paths were created for each region by randomly sampling start and goal locations and using A^* to plan optimal paths through the ground truth cost map.

8.3.3 MMP Backpropagation Results

In the experimental setup described previously, the neural network plays a key role in generating non-linear features that cannot be replicated by learning a cost function directly on the 5x5 pixel patches of the input image. Attempting to learn a cost function directly on the raw input patches failed, as shown in by the "Raw" bar in Figure 8.6. Here "Loss" refers to the fraction of the length

[2]Five classes: "paved road", "gravel road", "open field", "ditch", and "gully" were used to construct the ground truth cost map.

[3]Eliminating example inconsistencies required constructing a ground truth neural network. However, in practice when attempting to approximate human behavior, the optimal number and type of terrain classes would not be known. Therefore, the most prominent terrain classes in each region were labeled, as might be done in practice.

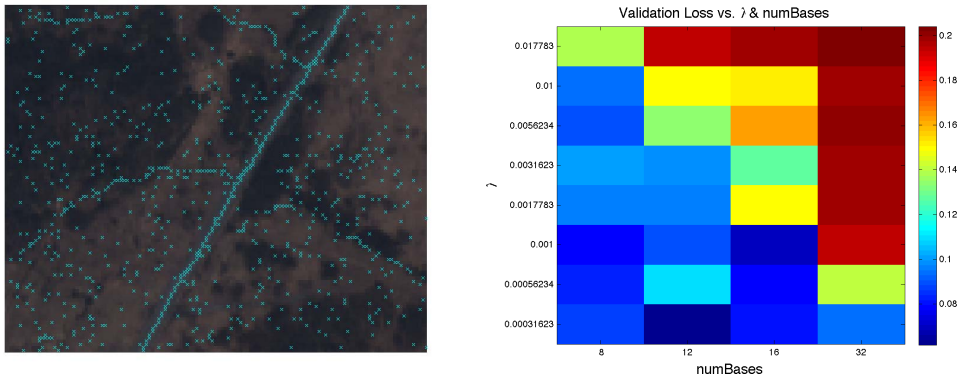


Figure 8.5: Randomly selected patches from the training map (left) were tried as an automatically generated auxiliary task for pre-training. Reasonable values for the regularization constant ($\lambda = 10^{-3.5}$) and number of bases (12) used to generate the sparse coding auxiliary task were selected to minimize the "pseudo-objective" (right) of training a maxent classifier to predict the ground truth terrain classes from the sparse codes.

of the example paths that were not matched by the minimum cost path returned by A^*. Figure 8.7 shows how loss corresponds to path-planning performance on the test region. Even on the training region examples it was only possible to match 20% of the length of the example paths using just the raw input features.

The "NN" bar in Figure 8.6 gives the results of adding a randomly initialized network between the raw input and the cost function. This network was the same size as the ground truth network, and it was trained by backpropagating gradients from the normalized MMP objective function. After backpropagation, the terrain classifications learned by the network allowed the system to approximately match most of the paths (Figure 8.7). Pre-training the network for the auxiliary terrain classification task (the labeled examples drawn from the training region) decreased the training time and improved the quality of the final result, as shown by the "PNN" bar in Figure 8.6. Pre-training the network with the automatically generated sparse coding task (the "SC" bar) proved no more useful than random initialization, highlighting the difficulty mentioned in Section 5.7 of converting a sparse coding module into part of a neural network. Due to computational limitations, the sparse coding module could not be differentiated directly as in Chapter 5, although that may produce superior results. For comparison purposes, the result of learning using the ground truth neural network classifications was included as "GT". As these classification maps would not be available in a real application, this result serves to bound the best performance that could be expected.

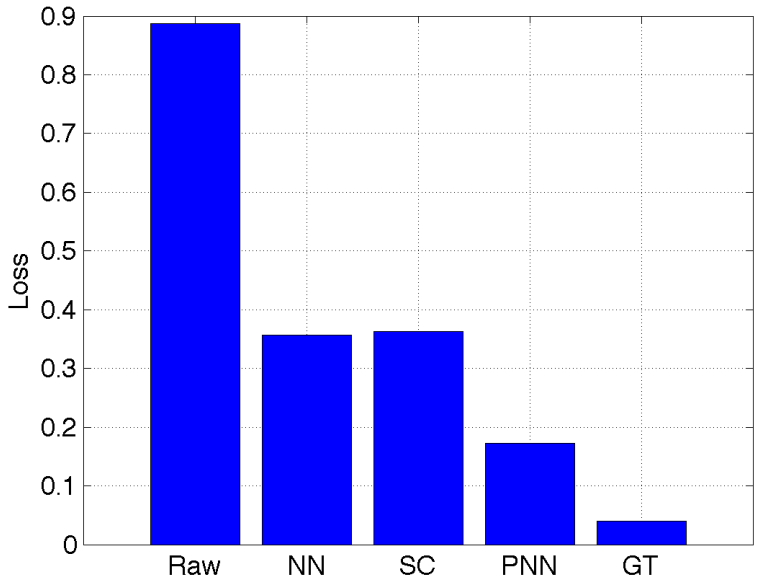


Figure 8.6: Test set loss of MMP backpropagation with and without pre-training. "Raw" is exponentiated functional gradient descent with linear regressors on 5x5 patches of satellite imagery. "NN" uses a randomly initialized neural network. "SC" uses a network that has been pre-trained on the automatically generated task of predicting the sparse codes for each patch. "PNN" employs a network that has been pre-trained on a human-designed terrain classification task before backpropagation. "GT" optimized only the cost function on the ground truth classifications using exponentiated functional gradient descent respectively. As the ground truth classifications would not be available in a real application, this bar is included to bound the best performance that could be expected. Results are reported in terms of example path loss, the fraction of the example path length that is not matched by the minimum cost path.

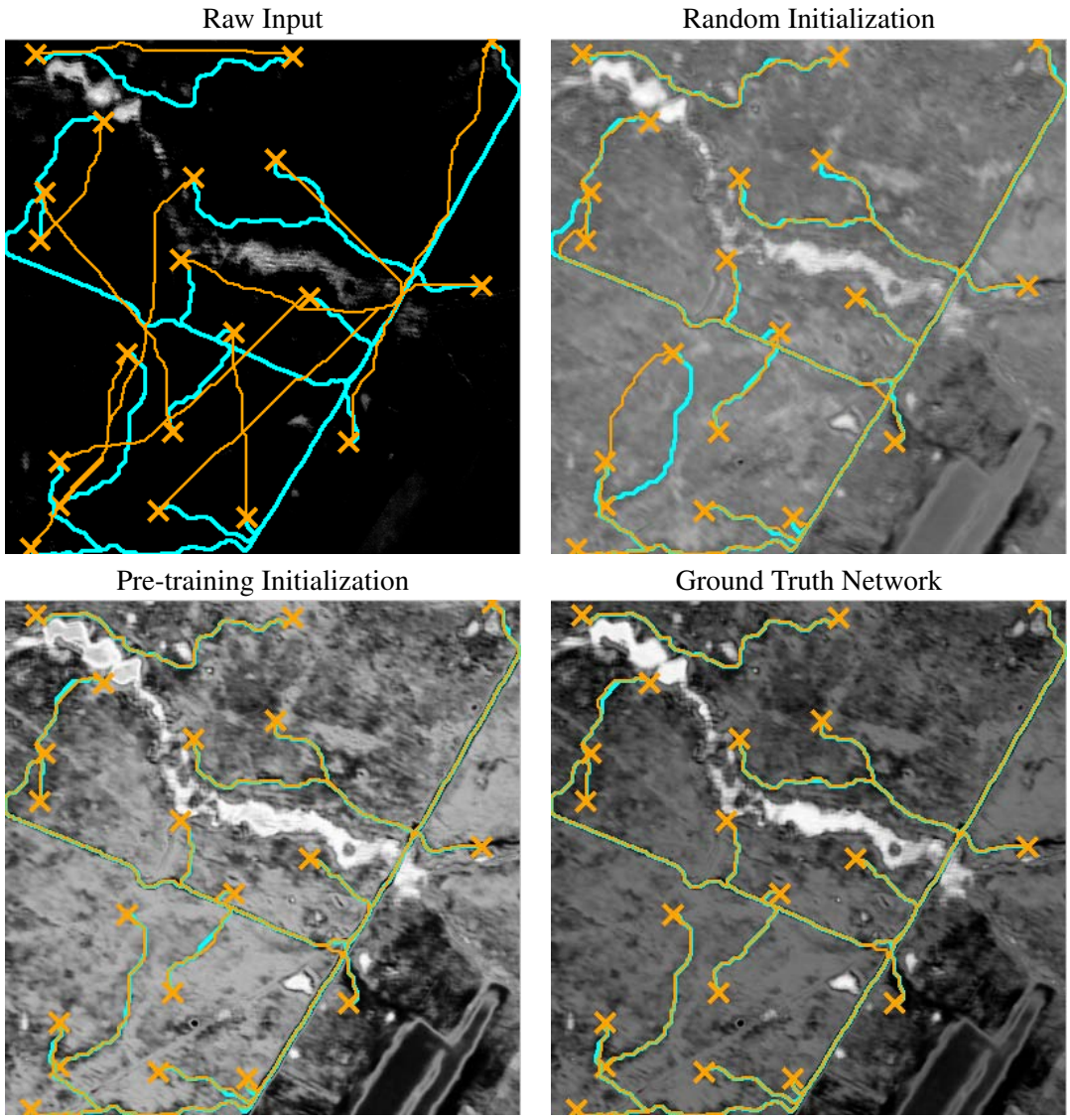


Figure 8.7: Cost-maps and minimum cost paths produced on the test region for various intermediate classification modules. Example paths are shown in cyan, minimum-cost paths are shown in orange, and start and goal locations are denoted by orange Xs. Top left: without an intermediate non-linear module, MMP is unable to match the example paths. Top right: gradient backpropagation on a randomly-initialized neural network produced useful features which matched most of the example paths. Bottom left: pre-training the neural network on a related terrain classification task improved performance and rivaled the result of using the ground-truth neural network classifications (Bottom right).

Bibliography

[1] L. Bottou and P. Gallinari, "A framework for the cooperation of learning algorithms," in *Advances in Neural Information Processing Systems*, D. Touretzky and R. Lippmann, Eds., vol. 3. Denver: Morgan Kaufmann, 1991. [Online]. Available: http://leon.bottou.org/papers/bottou-gallinari-90

[2] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop," in *Neural Networks: Tricks of the trade*, G. Orr and M. K., Eds. Springer, 1998.

[3] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. Cambridge, MA: MIT Press, 2007, pp. 153–160.

[4] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[5] N. Ratliff, J. D. Bagnell, and M. Zinkevich, "Maximum margin planning," in *International Conference on Machine Learning*, July 2006.

[6] N. Ratliff, D. Silver, and J. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, 2009.

[7] D. Bradley, R. Unnikrishnan, and J. Bagnell, "Vegetation detection for driving in complex environments," in *IEEE International Conference on Robotics and Automation*, April 2007.

[8] K. Sunkavalli, F. Romeiro, W. Matusik, T. Zickler, and H. Pfister, "What do color changes reveal about an outdoor scene?" *Computer Vision and Pattern Recognition, 2008.*, pp. 1 – 8, May 2008.

[9] J.-F. Lalonde, S. G. Narasimhan, and A. A. Efros, "What do the sun and the sky tell us about the camera?" Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-09-04, January 2009.

[10] D. Hoiem, A. Efros, and M. Hebert, "Closing the loop in scene interpretation," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008.*, pp. 1–8, 2008.

[11] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. to appear, 2009.

[12] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *ICML '07: Proceedings of the 24th international conference on Machine learning*. New York, NY, USA: ACM Press, 2007, pp. 473–480.

[13] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," *Proceedings of the 25th international conference on Machine learning*, pp. 1168–1175, 2008.

[14] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

[15] H. Mobahi, R. Collobert, and J. Weston, "Deep learning from temporal coherence in video," *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[16] D. Rumelhart, G. Hintont, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[17] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *In Advances in Neural Information Processing Systems 12*, vol. 12, 2000, pp. 512–518.

[18] J. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2001.

[19] Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte, "Convex neural networks," *Advances in Neural Information Processing Systems*, vol. 18, p. 123, 2006.

[20] D. M. Bradley and J. A. Bagnell, "Convex coding," in *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 2009.

[21] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: Transfer learning from unlabeled data," in *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007.

[22] D. Silver, J. Bagnell, and A. Stentz, "High performance outdoor navigation from overhead data using imitation learning," *Proceedings of Robotics Science and Systems*, Jan 2008.

[23] N. Ratliff, D. M. Bradley, J. A. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hofmann, Eds. Cambridge, MA: MIT Press, 2007.

[24] N. Ratliff, J. A. D. Bagnell, and S. Srinivasa, "Imitation learning for locomotion and manipulation," in *IEEE-RAS International Conference on Humanoid Robots*, November 2007.

[25] D. Silver, J. A. D. Bagnell, and A. T. Stentz, "Applied imitation learning for autonomous navigation in complex natural terrain," in *Field and Service Robotics*, July 2009.

[26] ——, "Perceptual interpretation for autonomous navigation through dynamic imitation learning," in *International Symposium of Robotics Research*, August 2009.

[27] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Twentieth International Conference on Machine Learning*, 2003.

[28] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[29] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[30] J. Kivinen and M. Warmuth, "Exponentiated gradient versus gradient descent for linear predictors," *Information and Computation*, pp. 1–63, 1997.

[31] L. G. Valiant, "A theory of the learnable," in *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1984, pp. 436–445.

[32] M. J. Kearns and U. V. Vazirani, *An Introduction To Computational Learning Theory*. MIT Press, 1994.

[33] T. M. Cover, "Geometrical and statistical properties of linear threshold decvices," Ph.D. dissertation, Stanford, May 1964.

[34] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation*, vol. 1, no. 1, pp. 151–160, 1989. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.1.151

[35] W. Maass, "Neural nets with superlinear vc-dimension," *Neural Computation*, vol. 6, pp. 877–884, 1994.

[36] Y. LeCun, B. Boser, J. Denker, and D. Henderson, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, Jan 1989.

[37] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. MIT Press, 1995.

[38] D. Erhan, P. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," *AI & Stat.'2009*, 2009.

[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, Mar 1994.

[40] Y. Bengio, "Learning deep architectures for ai," *Foundations & Trends in Mach. Learn., to appear*, 2009.

[41] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

[42] R. K. Ando and T. Zhang, "A framework for learning predictive structures from multiple tasks and unlabeled data," *Journal of Machine Learning Research*, pp. 1817–1853, November 2005. [Online]. Available: http://jmlr.csail.mit.edu/papers/v6/ando05a.html

[43] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

[44] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," *Proceedings of the 25th international conference on Machine learning*, pp. 1168–1175, 2008.

[45] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *International Conference on Machine Learning proceedings*, 2009.

[46] Champeny-Bares, L. S. Coppersmith, and K. Dowling, "The terregator mobile robot," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-93-03, May 1991.

[47] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 1, pp. 362 – 373, May 1988.

[48] K. E. Olin and D. Y. Tseng, "Autonomous cross-country navigation: An integrated perception and planning system," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 6, no. 4, pp. 16–30, 1991.

[49] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100 – 107, Jul 1968.

[50] A. T. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," in *Proceedings 1995 IEEE/RSJ International Conference On Intelligent Robotic Systems (IROS '95)*, vol. 1, August 1995, pp. 425 – 432.

[51] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. Vallidis, and R. Warner, "Toward reliable off road autonomous vehicles operating in challenging environments," *International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 449–483, 2006.

[52] D. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1*, D. Touretzky, Ed. Morgan Kaufmann, 1989.

[53] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-road obstacle avoidance through end-to-end learning," in *Advances in Neural Information Processing Systems (NIPS 2005)*. MIT Press, 2005.

[54] C. Wellington, A. Courville, and A. T. Stentz, "A generative model of terrain for autonomous navigation in vegetation," *The International Journal of Robotics Research*, vol. 25, no. 1, pp. 1287 – 1304, December 2006.

[55] B. Sofman, E. L. Ratliff, J. A. D. Bagnell, J. Cole, N. Vandapel, and A. T. Stentz, "Improving robot navigation through self-supervised online learning," *Journal of Field Robotics*, vol. 23, no. 1, December 2006.

[56] N. Vandapel and M. Hebert, "Natural terrain classification using 3-d ladar data," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2004.

[57] J.-F. Lalonde, N. Vandapel, D. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," *Journal of Field Robotics*, vol. 23, no. 1, pp. 839 – 861, November 2006.

[58] D. Bradley, S. Thayer, A. T. Stentz, and P. Rander, "Vegetation detection for mobile robot navigation," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-04-12, February 2004.

[59] J. Macedo, R. Manduchi, and L. Matthies, "Ladar-based discrimination of grass from obstacles for autonomous navigation," in *Proc. of the International Symposium on Experimental Robotics*, Honolulu, HA, December 2000.

[60] R. N. Clark, G. A. Swayze, K. E. Livo, R. F. Kokaly, S. J. Sutley, J. B. Dalton, R. R. McDougal, and C. Gent, "Imaging spectroscopy: Earth and planetary remote sensing with the usgs tetracorder and expert systems," *J. Geophys. Res.*, December 2003.

[61] R. E. Crippen, "Calculating the vegetation index faster," *Remote Sensing of Environment*, vol. 34, pp. 71–73, 1990.

[62] A. R. Huete, "A soil adjusted vegetation index (savi)," *Remote Sensing of Environment*, vol. 25, pp. 295–309, 1988.

[63] C. F. Jordan, "Derivation of leaf area index from quality measurements of light on the forest floor," *Ecology*, vol. 50, pp. 663–666, 1969.

[64] R. J. Kauth and G. S. Thomas, "The tasseled cap - a graphic description of the spectral-temporal development of agricultural crops as seen by landsat," in *LARS: Proceedings of the Symposium on Machine Processing of Remotely Sensed Data*, West Lafayette, IN, 1976, pp. 4B–14–4B–51.

[65] F. J. Kreigler, W. A. Malila, R. Nalepka, and W. Richardson, "Preprocessing transformations and their effects on multispectral recognition," in *Proc. of the Sixth International Symposium on Remote Sensing of Environment*, Ann Arbor, MI, 1969, pp. 97–131.

[66] R. Willstatter and A. Stoll, *Utersuchungenuber Chlorophyll*. Berlin: Springer, 1913.

[67] A. J. Richardson and C. L. Wiegand, "Distinguishing vegetation from soil background information," *Photogrammetric Engineering and Remote Sensing*, vol. 43, pp. 1541–1552, 1977.

[68] L. Matthies, A. Kelly, T. Litwin, and G. Tharp, "Obstacle detection for unmanned ground vehicles: A progress report," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 1996, pp. 475–486. [Online]. Available: citeseer.ist.psu.edu/matthies95obstacle.html

[69] R. Manduchi, A. Castano, A. Talukder, and L. Matthies, "Obstacle detection and terrain classification for autonomous off-road navigation," *Autonomous Robots*, vol. 18, pp. 81–102, 2005.

[70] C. A. Shull, "A spectrophotometric study of the reflection of light from leaf surfaces," *Bot. Gazette*, vol. 87, no. 5, pp. 583–607, 1929.

[71] J. Qi, A. Chehbouni, A. R. Huete, Y. H. Kerr, and S. Sorooshian, "A modified soil adjusted vegetation index: Msavi," *Remote Sensing of Environment*, vol. 48, pp. 119–126, 1994.

[72] C. Ünsalan and K. L. Boyer, "Linearized vegetation indices based on a formal statistical framework," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 7, pp. 1575–1585, 2004.

[73] R. N. Clark, G. A. Swayze, R. Wise, K. E. Livo, T. M. Hoefen, R. F. Kokaly, and S. J. Sutley, "USGS digital spectral library splib05a, USGS open file report 03-395," 2003.

[74] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000.

[75] H. Zhang, A. Berg, M. Maire, and J. Malik, "Svm-knn: Discriminative nearest neighbor classification for visual category recognition," *2006 IEEE Computer Society Conference on Computer . . .* , Jan 2006.

[76] S. Bickel, C. Sawade, and T. Scheffer, "Transfer learning by distribution matching for targeted advertising," in *Advances in Neural Information Processing Systems 21, Proceedings of the 22nd Annual Conference on Neural Information Processing Systems*. Vancouver, Canada: MIT, 2009.

[77] D. M. Bradley and J. A. Bagnell, "Differentiable sparse coding," in *Neural Information Processing Systems 22*, 2009.

[78] J. A. Tropp, "Algorithms for simultaneous sparse approximation: part ii: Convex relaxation," *Signal Process.*, vol. 86, no. 3, pp. 589–602, 2006.

[79] B. Olshausen and D. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision Research*, 1997.

[80] Y. Karklin and M. S. Lewicki, "A hierarchical bayesian model for learning non-linear statistical regularities in non-stationary natural signals," *Neural Computation*, vol. 17, no. 2, pp. 397–423, 2005.

[81] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, 2009.

[82] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Supervised dictionary learning," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009.

[83] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. Cambridge, MA: MIT Press, 2007.

[84] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *Journal of Machine Learning Research*, vol. 6, pp. 1705–1749, 2005.

[85] J. Mairal, G. Sapiro, and M. Elad, "Learning multiscale sparse representations for image and video restoration," *Multiscale Modeling & Simulation*, vol. 7, no. 1, pp. 214–241, 2008.

[86] M. Brand, "Pattern discovery via entropy minimization," in *AISTATS 99*, 1999.

[87] M. Shashanka, B. Raj, and P. Smaragdis, "Sparse overcomplete latent variable decomposition of counts data," in *NIPS*, 2007.

[88] G. Besnerais, J. Bercher, and G. Demoment, "A new look at entropy for solving linear inverse problems," *IEEE Trans. on Information Theory*, vol. 45, no. 5, pp. 1565–1578, July 1999.

[89] D. Widder, *Advanced Calculus*, 2nd ed. Dover Publications, 1989.

[90] G. Strang, *Linear Algebra and its Applications*, 3rd ed. Harcourt Brace Jovanovich College Publishers, 1988.

[91] R. Duda, P. Hart, and D. Stork, *Pattern classification*. Wiley New York, 2001.

[92] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[93] K. Nigam, J. Lafferty, and A. McCallum, "Using maximum entropy for text classification," 1999. [Online]. Available: citeseer.ist.psu.edu/article/nigam99using.html

[94] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*. Washington, DC, USA: IEEE Computer Society, 2003, p. 958.

[95] D. M. Blei and J. D. McAuliffe, "Supervised topic models," in *NIPS 19*, 2007.

[96] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the ACL*, 2005, pp. 115–124.

[97] R. Salakhutdinov and G. Hinton, "Semantic hashing," in *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007.

[98] M. Elad and M. Aharon, "Image denoising via learned dictionaries and sparse representation," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 895–900.

[99] Y. Karklin and M. Lewicki, "Emergence of complex cell properties by learning to generalize in natural scenes," *Nature*, vol. 457, pp. 83–86, November 2009.

[100] J. Yang, K. Yu, and T. Huang, "Supervised translation-invariant sparse coding," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[101] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[102] K. Yu, "Personal communications," May 2910.

[103] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," in *NIPS*, December 2006.

[104] G. Obozinski, B. Taskar, and M. Jordan, "Multi-task feature selection," in *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, June 2006.

[105] J. A. Tropp, A. C. Gilbert, and M. J. Strauss, "Algorithms for simultaneous sparse approximation: part i: Greedy pursuit," *Signal Process.*, vol. 86, no. 3, pp. 572–588, 2006.

[106] F. Bach, J. Mairal, and J. Ponce, "Convex sparse matrix factorizations," *CoRR*, vol. abs/0812.1869, 2008.

[107] C. Ding, X. He, and H. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proceedings of the SIAM Data Mining Conference*, 2005.

[108] D. Lashkari and P. Golland, "Convex clustering with exemplar-based models," in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA: MIT Press, 2008, pp. 825–832.

[109] S. Nowozin and G. Bakir, "A decoupled approach to exemplar-based unsupervised learning," in *ICML '08: Proceedings of the 25th international conference on Machine learning*. New York, NY, USA: ACM, 2008, pp. 704–711.

[110] J. Borwein and A. Lewis, *Convex Analysis and Nonlinear Optimization*, 2nd ed. CMS-Springer Books, 2005.

[111] J. Tropp, "Algorithms for simultaneous sparse approximation. part ii: Convex relaxation," *Signal Processing*, Jan 2006.

[112] R. Rifkin and R. Lippert, "Value regularization and fenchel duality," *The Journal of Machine Learning Research*, vol. 8, pp. 441–479, 2007.

[113] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *Proc. IROS*, 2009.

[114] K. W. Fitzpatrick, R. C. Coulter, and H. M. Pangels, "Real-time prediction and management of food product demand," US Patent 6 842 719, 2005.

[115] N. Shor, K. Kiwiel, and A. Ruszcayǹski, "Minimization methods for non-differentiable functions," *Springer-Verlag New York, Inc. New York, NY, USA*, Jan 1985.

[116] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field d* algorithm," *Journal of Field Robotics*, Jan 2006.

[117] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *International Conference on Machine Learning*, 2009.